Lecture Notes in Computer Science 1853

Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Springer
Berlin
Heidelberg
New York
Barcelona
Hong Kong
London
Milan
Paris
Singapore
Tokyo

Ugo Montanari José D.P. Rolim
Emo Welzl (Eds.)

Automata, Languages and Programming

27th International Colloquium, ICALP 2000
Geneva, Switzerland, July 9-15, 2000
Proceedings

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Ugo Montanari
University of Pisa, Department of Computer Sciences
Corso Italia, 40, 56125 Pisa, Italy
E-mail: ugo@di.unipi.it

José D.P. Rolim
University of Geneva, Center for Computer Sciences
24, Rue Général Dufour, 1211 Geneva 4, Switzerland
E-mail: Jose.Rolim@cui.unige.ch

Emo Welzl
ETH Zurich, Department of Computer Sciences
8092 Zurich, Switzerland
E-mail: emo@inf.ethz.ch

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Automata, languages and programming : 27th international colloquium ; proceedings / ICALP 2000, Geneva, Switzerland, July 9 - 15, 2000. Ugo Montanari . . . (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1853)
ISBN 3-540-67715-1

CR Subject Classification (1998): F, D, C.2-3, G.1-2

ISSN 0302-9743
ISBN 3-540-67715-1 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag is a company in the BertelsmannSpringer publishing group.
© Springer-Verlag Berlin Heidelberg 2000
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Steingräber Satztechnik GmbH, Heidelberg
Printed on acid-free paper SPIN: 10722141 06/3142 5 4 3 2 1 0

Preface

This volume contains the papers presented at the *27th International Colloquium on Automata, Languages and Programming* (ICALP 2000), which took place at the University of Geneva, Switzerland, July 9–15, 2000.

The volume contains 69 contributed papers, selected by the two program committees from 196 extended abstracts submitted in response to the call for papers: 42 from 131 submissions for track A (Algorithms, Automata, Complexity, and Games) and 27 from 65 submissions for track B (Logic, Semantics, and Theory of Programming). Moreover, the volume includes abstracts of a plenary lecture by Richard Karp and of invited lectures by Samson Abramsky, Andrei Broder, Gregor Engels, Oded Goldreich, Roberto Gorrieri, Johan Håstad, Zohar Manna, and Kurt Mehlhorn.

The program committees decided to split the EATCS best paper awards among the following three contributions: *Deterministic algorithms for k-SAT based on covering codes and local search*, by Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, and Uwe Schöning, *Reasoning about idealized Algol using regular languages*, by Dan R. Ghica and Guy McCusker, and *An optimal minimum spanning tree algorithm*, by Seth Pettie and Vijaya Ramachandran.

The best student paper award for track A was given to *Clique is hard to approximate within* $n^{1-o(1)}$, by Lars Engebretsen and Jonas Holmerin, and for track B to *On deciding if deterministic Rabin language is in Büchi class*, by Tomasz Fryderyk Urbanski.

We thank all of the authors who submitted papers, our invited speakers, the external referees we consulted, and the members of the program committees, who were:

Track A
- Peter Bro Miltersen, U. Aarhus
- Harry Buhrman, CWI Amsterdam
- Martin Dietzfelbinger, TU Ilmenau
- Afonso Ferreira, Inria Sophia Ant.
- Marcos Kiwi, U. Chile
- Jens Lagergren, KTH Stockholm
- Gheorghe Paun, Romanian Acad.
- Günter Rote, FU Berlin
- Ronitt Rubinfeld, NECI
- Amin Shokrollahi, Bell Labs
- Luca Trevisan, Columbia U.
- Serge Vaudenay, EPF Lausanne
- Emo Welzl, *Chair*, ETH Zürich
- Uri Zwick, Tel Aviv U.

Track B
- Rajeev Alur, U. Pennsylvania
- Rance Cleaveland, Stony Brook
- Pierpaolo Degano, U. Pisa
- Jose Fiadeiro, U. Lisbon
- Andy Gordon, Microsoft Cambridge
- Orna Grumberg, Technion Haifa
- Claude Kirchner, INRIA Nancy
- Ugo Montanari, *Chair*, U. Pisa
- Mogens Nielsen, U. Aarhus
- Catuscia Palamidessi, Penn. State
- Joachim Parrow, KTH Stockholm
- Edmund Robinson, QMW London
- Jan Rutten, CWI Amsterdam
- Jan Vitek, U. Geneva
- Martin Wirsing, LMU Munich
- Pierre Wolper, U. Liege

We gratefully acknowledge support from the Swiss National Science Foundation, from the computer science department of the University of Geneva, from the European agency INTAS, and from the EATCS. Finally, we would like to thank the local arrangement committee members – Olivier Powell, Frédéric Schütz, Danuta Sosnowska, and Thierry Zwissig – and Germaine Gusthiot for the secretarial support.

July 2000

Emo Welzl, Track A Program Chair
Ugo Montanari, Track B Program Chair
José D. P. Rolim, General Chair

Referees

Slim Abdennadher
Farid Ablayev
Luca Aceto
Susanne Albers
Eric Allender
Noga Alon
Eitan Altman
Rajeev Alur
Christoph Ambühl
Andris Ambainis
Artur Andrzejak
Farhad Arbab
Stefan Arnborg
Alfons Avermiddig
Yossi Azar
Evripidis Bampis
Nuno Barreiro
Lali Barriere
David Mix Barrington
Klaus Georg Barthelmann
Olivier Baudron
Martin Beaudry
Bruno Beauquier
Marco Bellia
Michael Bender
Nick Benton
Veronique Benzaken
Daniel Bleichenbacher
Peter van Emde Boas
Alexander Bockmayr
Chiara Bodei
Hans Bodlaender
Maria Bonet
Marcello Bonsangue
Michele Boreale
Vincent Bouchitté
Olivier Bournez
Peter Braß
Roberto Bruni
Peter Buergisser
Nadia Busi
Edson Caceres
Luis Caires
Sergio Campos
Giuseppe Castagna
Sébastien Choplin
Horatiu Cirstea
Rance Cleaveland
Andrea Clementi
Peter Clote
Philippe Codognet
Johanne Cohen
Jean-Sébastien Coron
Ricardo Corrêa
David Coudert
Ronald Cramer
Pierluigi Crescenzi
Felipe Cucker
Mads Dam
Ivan Damgaard
Olivier Danvy
Alain Darte
Nicola Rocco De
Pierpaolo Degano
Hartog Jerry Den
Theodosis Dimitrakos
Yevgeniy Dodis
Gilles Dowek
Frank Drewes
Jérôme Durand-Lose
Christoph Durr
Bruno Dutertre
Norbert Eisinger
Joost Engelfriet
Adam Eppendahl
Lea Epstein
Zoltan Esik
Javier Esparza
Kousha Etessami
Jean-Marc Fédou

Uri Feige
Joan Feigenbaum
Sándor Fekete
Michael Fellows
Stefan Felsner
Gianluigi Ferrari
Esteban Feuerstein
Jose Fiadeiro
Lance Fortnow
Cedric Fournet
Pierre Fraigniaud
Marcelo Frias
Roy Friedman
Bernd Gärtner
Fabio Gadducci
Anna Gal
Jérôme Galtier
Juan Garay
Max Garzon
Bill Gasarch
Cyril Gavoille
Rosario Gennaro
Giorgio Ghelli
Pablo Giambiagi
Leslie Goldberg
Mikael Goldmann
Andy Gordon
Roberto Gorrieri
Louis Granboulan
Radu Grosu
Orna Grumberg
Peter Grunwald
Sudipto Guha
Venkatesan Guruswami
Shai Halevi
Michael T. Hallett
Mikael Hammar
Therese Hardin
Laura Heinrich-Litan
Matthew Hennessy
Jesper Gulmann Henriksen
Miki Hermann
W. Hesselink
Clemens Heuberger
Jaap-Henk Hoepman
Frank Hoffmann
Michael Hoffmann
Johan Holmerin
Furio Honsell
Hendrik Jan Hoogeboom
Jacob Howe
Peter Hoyer
Juraj Hromkovic
Sebastian Hunt
Hans Huttel
Johan Håstad
Sandy Irani
Lars Ivansson
Sanjay Jain
Peter Jancar
David Janin
Klaus Jansen
Mark Jerrum
Tania Jimenez
Öjvind Johansson
Bengt Jonsson
Laurent Juban
Gabriel Juhas
Sandrine Julia
Marcin Jurdzinski
Astrid Kaffanke
Viggo Kann
Juhani Karhumäki
Sagi Katz
Sanjeev Khanna
Joe Kilian
Claude Kirchner
Alexander Knapp
Christian Knauer
Ulrich Kortenkamp
Piotr Kosiuczenko
Jürgen Koslowski
Ingolf Krüger
Klaus Kriegel
Michael Krivelevich
Danny Krizanc
Hillel Kugler
Ravi Kumar

Gabriel Kuper
Orna Kupferman
Yassine Lakhnech
Francois Lamarche
Kim G. Larsen
Isabelle Guérin Lassous
Marina Lenisa
Stefano Leonardi
Francesca Levi
Paul Levy
Sebastien Limet
Huimin Lin
Luigi Liquori
Björn Lisper
Martin Loebl
Antonia Lopes
Jack Lutz
Andy Mück
Philip MacKenzie
Frederic Magniez
Pasquale Malacaria
Tal Malkin
Karina Marcus
Narciso Marti-Oliet
Bruno Martin
Simone Martini
GianCarlo Mauri
Elvira Mayordomo
Richard Mayr
Robert McNaughton
Klaus Meer
Lutz Meißner
Dieter van Melkebeek
Stephan Merz
Hermann Miki
Dale Miller
Joseph Mitchell
Michael Mitzenmacher
Faron Moller
Nicole Morawe
Matthew Morley
Till Mossakowski
Peter D. Mosses
Dalit Naor
Uwe Nestmann
Joachim Niehren
Mogens Nielsen
Flemming Nielson
Karl-Heinz Niggl
Isabel Nunes
Peter O'Hearn
Mitsu Ogihara
Vadim Olshevsky
Luke Ong
Andre Osterloh
Rafi Ostrovsky
Catuscia Palamidessi
Daniel Panario
Joachim Parrow
Malacaria Pasquale
Christian Storm Pedersen
Dino Pedreschi
Samuele Pedroni
Andrezj Pelć
Paolo Penna
Stephane Perennes
Adriano Peron
Antoine Petit
Birgit Pfitzmann
Benny Pinkas
Andrew Pitts
Dennis Pixton
John Pliam
David Pointcheval
Katerina Pokozy
Carl Pomerance
Corrado Priami
Rosario Pugliese
Tal Rabin
Ivan Rapaport
Anna Redz
Oded Regev
Omer Reingold
Arend Rensink

Jürgen Richter-Gebert
Søren Riis
Edmund Robinson
Mario Rodriguez-Artalejo
Dana Ron
Roni Rosner
Francesca Rossi
Alex Russell
Jan Rutten
Alex Samorodinitsky
Tomas Sander
Davide Sangiorgi
Miklos Santha
Vladimiro Sassone
Uwe Schöning
Marcus Schaefer
Berry Schoenmakers
K.U. Schulz
Andreas Schulz
Philip Scott
Francesca Scozzari
Sebastian Seiber
Maria Jose Serna
Peter Sewell
Detlev Sieling
Riccardo Silvestri
Alex Simpson
Alistair Sinclair
Seppo Sippu
D. Sivakumar
Carl Smith
Warren Smith
József Solymosi
Dan Spielman
Jiri Srba
Aravind Srinivasan
Bernhard von Stengel
Bernd Sturmfels
Harald Störrle
Madhu Sudan
Amnon Ta-Shma
Denis Therien
P.S. Thiagarajan
Lothar Thiele
Hayo Thielecke
Thomas Thierauf
Wolfgang Thomas
Mikkel Thorup
Simone Tini
Jacobo Toran
Leen Torenvliet
John Tromp
Daniele Turi
Christophe Tymen
Staffan Ulfberg
Christian Urban
Tomas Uribe
Mark Van der Zwaag
Moshe Vardi
Vasco Vasconcelos
Santosh Vempala
Betti Venneri
Bjorn Victor
Paul Vitanyi
Jan Vitek
Uli Wagner
Michal Walicki
John Watrous
Ingo Wegener
Pascal Weil
Carola Wenk
Benjamin Werner
Susanne Wetzel
Peter Widmayer
Thomas Wilke
Glynn Winskel
Martin Wirsing
Ronald de Wolf
Pierre Wolper
James Worrel
Wang Yi
Mingsheng Ying
Domenico Zambella
Francis Zane
Leonid Zosin

Table of Contents

Invited Talk: 1
Game Semantics: Achievements and Prospects
Samson Abramsky

Clique Is Hard to Approximate within $n^{1-o(1)}$ 2
Lars Engebretsen, Jonas Holmerin

Approximating the Independence Number and the Chromatic Number in Expected Polynomial Time 13
Michael Krivelevich, Van H. Vu

Closed Types as a Simple Approach to Safe Imperative Multi-stage Programing 25
Cristiano Calcagno, Eugenio Moggi, Walid Taha

A Statically Allocated Parallel Functional Language 37
Alan Mycroft, Richard Sharp

An Optimal Minimum Spanning Tree Algorithm 49
Seth Pettie, Vijaya Ramachandran

Improved Shortest Paths on the Word RAM 61
Torben Hagerup

Improved Algorithms for Finding Level Ancestors in Dynamic Trees 73
Stephen Alstrup, Jacob Holm

Lax Logical Relations 85
Gordon Plotkin, John Power, Donald Sannella, Robert Tennent

Reasoning about Idealized Algol Using Regular Languages 103
Dan R. Ghica, Guy McCusker

The Measurement Process in Domain Theory 116
Keye Martin

Invited Talk: 127
Graph Transformation as a Conceptual and Formal Framework for System Modeling and Model Evolution
Gregor Engels, Reiko Heckel

Monotone Proofs of the Pigeon Hole Principle 151
Albert Atserias, Nicola Galesi, Ricard Gavaldà

Fully-Abstract Statecharts Semantics via Intuitionistic Kripke Models 163
Gerald Lüttgen, Michael Mendler

Algebraic Models for Contextual Nets 175
Roberto Bruni, Vladimiro Sassone

Asymptotically Optimal Bounds for OBDDs and the Solution
of Some Basic OBDD Problems 187
Beate Bollig, Ingo Wegener

Measures of Nondeterminism in Finite Automata 199
Juraj Hromkovič, Juhani Karhumäki, Hartmut Klauck, Georg Schnitger, Sebastian Seibert

LTL Is Expressively Complete for Mazurkiewicz Traces 211
Volker Diekert, Paul Gastin

An Automata-Theoretic Completeness Proof for Interval Temporal Logic . 223
Ben C. Moszkowski

Invited Talk: 235
Which NP-Hard Optimization Problems Admit
Non-trivial Efficient Approximation Algorithms?
Johan Håstad

Deterministic Algorithms for k-SAT Based on Covering Codes
and Local Search 236
Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Uwe Schöning

Closest Vectors, Successive Minima, and Dual HKZ-Bases of Lattices 248
Johannes Blömer

Variable Independence, Quantifier Elimination,
and Constraint Representations 260
Leonid Libkin

Constraint Satisfaction Problems and Finite Algebras 272
Andrei A. Bulatov, Andrei A. Krokhin, Peter Jeavons

An Optimal Online Algorithm
for Bounded Space Variable-Sized Bin Packing 283
Steven S. Seiden

Resource Augmentation for Online Bounded Space Bin Packing 296
János Csirik, Gerhard J. Woeginger

Optimal Projective Algorithms for the List Update Problem 305
Christoph Ambühl, Bernd Gärtner, Bernhard von Stengel

Efficient Verification Algorithms for One-Counter Processes 317
Antonín Kučera

On the Complexity of Bisimulation Problems for Basic Parallel Processes . 329
Richard Mayr

Decidable First-Order Transition Logics for PA-Processes 342
Denis Lugiez, Philippe Schnoebelen

Invited Talk: 354
Non Interference for the Analysis of Cryptographic Protocols
Riccardo Focardi, Roberto Gorrieri, Fabio Martinelli

Average Bit-Complexity of Euclidean Algorithms 373
Ali Akhavi, Brigitte Vallée

Planar Maps and Airy Phenomena 388
Cyril Banderier, Philippe Flajolet, Gilles Schaeffer, Michèle Soria

Analysing Input/Output-Capabilities of Mobile Processes
with a Generic Type System 403
Barbara König

Information Flow vs. Resource Access in the Asynchronous Pi-Calculus ... 415
Matthew Hennessy, James Riely

Award Talk: 428
The Genomics Revolution and Its Challenges for Algorithmic Research
Richard M. Karp

Invited Talk: 429
Alternating the Temporal Picture for Safety
Zohar Manna and Henny B. Sipma

Necessary and Sufficient Assumptions for Non-interactive
Zero-Knowledge Proofs of Knowledge for All NP Relations 451
Alfredo De Santis, Giovanni Di Crescenzo, Giuseppe Persiano

Fast Verification of Any Remote Procedure Call:
Short Witness-Indistinguishable One-Round Proofs for NP 463
William Aiello, Sandeep Bhatt, Rafail Ostrovsky, S. Raj Rajagopalan

A New Unfolding Approach to LTL Model Checking 475
Javier Esparza, Keijo Heljanko

Reasoning about Message Passing in Finite State Environments 487
B. Meenakshi, R. Ramanujam

Extended Notions of Security for Multicast Public Key Cryptosystems 499
Olivier Baudron, David Pointcheval, Jacques Stern

One-Round Secure Computation and Secure Autonomous Mobile Agents . 512
Christian Cachin, Jan Camenisch, Joe Kilian, Joy Müller

Round-Optimal and Abuse Free Optimistic Multi-party Contract Signing . 524
Birgit Baum-Waidner, Michael Waidner

On the Centralizer of a Finite Set 536
Juhani Karhumäki, Ion Petre

On the Power of Tree-Walking Automata 547
Frank Neven, Thomas Schwentick

Determinization of Transducers over Infinite Words 561
Marie-Pierre Béal, Olivier Carton

Invited Talk: 571
Constraint Programming and Graph Algorithms
Kurt Mehlhorn

Scalable Secure Storage when Half the System Is Faulty 576
Noga Alon, Haim Kaplan, Michael Krivelevich, Dahlia Malkhi, Julien Stern

Generating Partial and Multiple Transversals of a Hypergraph 588
Endre Boros, Vladimir Gurvich, Leonid Khachiyan, Kazuhisa Makino

Revisiting the Correspondence between Cut Elimination and Normalisation 600
José Espírito Santo

Negation Elimination from Simple Equational Formulae 612
Reinhard Pichler

Hardness of Set Cover with Intersection 1 624
V.S. Anil Kumar, Sunil Arya, H. Ramesh

Strong Inapproximability of the Basic k-Spanner Problem 636
Michael Elkin, David Peleg

Infinite Series-Parallel Posets: Logic and Languages 648
Dietrich Kuske

On Deciding if Deterministic Rabin Language Is in Büchi Class 663
Tomasz Fryderyk Urbański

On Message Sequence Graphs and Finitely Generated Regular MSC Languages 675
Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, P.S. Thiagarajan

Invited Talk: 687
Pseudorandomness
Oded Goldreich

A Bound on the Capacity of Backoff and Acknowledgement-Based Protocols 705
Leslie Ann Goldberg, Mark Jerrum, Sampath Kannan, Mike Paterson

Deterministic Radio Broadcasting 717
Bogdan S. Chlebus, Leszek Gąsieniec, Anna Östlin, John Michael Robson

An ω-Complete Equational Specification of Interleaving 729
W.J. Fokkink, S.P. Luttik

A Complete Axiomatization for Observational Congruence of Prioritized Finite-State Behaviors 744
Mario Bravetti, Roberto Gorrieri

Tight Size Bounds for Packet Headers in Narrow Meshes 756
Micah Adler, Faith Fich, Leslie Ann Goldberg, Mike Paterson

Wavelength Assignment Problem on All-Optical Networks with k Fibres per Link 768
Luciano Margara, Janos Simon

On the Logical Characterisation of Performability Properties 780
Christel Baier, Boudewijn Haverkort, Holger Hermanns, Joost-Pieter Katoen

On the Representation of Timed Polyhedra 793
Olivier Bournez, Oded Maler

Invited Talk: 808
Min-wise Independent Permutations: Theory and Practice
Andrei Z. Broder

Testing Acyclicity of Directed Graphs in Sublinear Time 809
Michael A. Bender, Dana Ron

Computing the Girth of a Planar Graph 821
Hristo N. Djidjev

Lower Bounds Are Not Easier over the Reals: Inside PH 832
Hervé Fournier, Pascal Koiran

Unlearning Helps 844
Ganesh Baliga, John Case, Wolfgang Merkle, Frank Stephan

Fast Approximation Schemes for Euclidean Multi-connectivity Problems .. 856
Artur Czumaj, Andrzej Lingas

Approximate TSP in Graphs with Forbidden Minors 869
Michelangelo Grigni

Polynomial Time Approximation Schemes
for General Multiprocessor Job Shop Scheduling 878
Klaus Jansen, Lorant Porkolab

The Many Faces of a Translation 890
Pierre McKenzie, Thomas Schwentick, Denis Thérien, Heribert Vollmer

Gales and the Constructive Dimension of Individual Sequences 902
Jack H. Lutz

The Global Power of Additional Queries to p-Random Oracles 914
Wolfgang Merkle

Homogenization and the Polynomial Calculus 926
Josh Buresh-Oppenheim, Matt Clegg, Russell Impagliazzo, Toniann Pitassi

Author Index 939

Game Semantics: Achievements and Prospects

Samson Abramsky

Department of Computer Science
University of Edinburgh
http://www.dcs.ed.ac.uk/home/samson/

Abstract. Game-theoretic ideas have been used to model a wide range of computational notions over the past few years. This has led to some striking results of a foundational character, relating in porticular to definability and full abstraction. The first applications of these ideas, to program analysis and verification, have also begun to appear. We shall give an overview of what has been achieved, and try to map out some objectives for future research.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, p. 1, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Clique Is Hard to Approximate within $n^{1-o(1)}$

Lars Engebretsen and Jonas Holmerin

Royal Institute of Technology
Department of Numerical Analysis and Computing Science
SE-100 44 Stockholm, SWEDEN
Fax: +46 8 790 09 30
{enge,joho}@nada.kth.se

Abstract. It was previously known that Max Clique cannot be approximated in polynomial time within $n^{1-\epsilon}$, for any constant $\epsilon > 0$, unless $\mathbf{NP} = \mathbf{ZPP}$. In this paper, we extend the reductions used to prove this result and combine the extended reductions with a recent result of Samorodnitsky and Trevisan to show that clique cannot be approximated within $n^{1-O(1/\sqrt{\log\log n})}$ unless $\mathbf{NP} \subseteq \mathbf{ZPTIME}(2^{O(\log n(\log\log n)^{3/2})})$.

1 Introduction

The Max Clique problem, i.e., the problem of finding in a graph $G = (V, E)$ the largest possible subset C of the vertices in V such that every vertex in C has edges to all other vertices in C, is a well-known combinatorial optimization problem. The decision version of Max Clique was one of the problems proven to be **NP**-complete in Karp's original paper on **NP**-completeness [10], which means that we cannot hope to solve Max Clique efficiently, at least not if we want an exact solution. Thus, attention has turned to algorithms producing solutions which are at most some factor from the optimum value. It is trivial to approximate Max Clique in a graph with n vertices within n—just pick any vertex as the clique—and Boppana and Halldórsson [5] have shown that Max Clique can be approximated within $O(n/\log^2 n)$ in polynomial time. It is an astonishing, and unfortunate, result that it is hard to do substantially better than this. In fact, the Max Clique problem cannot be approximated within $n^{1-\epsilon}$, for any constant $\epsilon > 0$, unless $\mathbf{NP} = \mathbf{ZPP}$. The first to explore the possibility of proving strong lower bounds on the approximability of Max Clique were Feige et al. [8], who proved a connection between Max Clique and probabilistic proof systems. Their reduction was then improved independently by Bellare, Goldreich, and Sudan [3] and Zuckerman [12]. As the final link in the chain, Håstad [9] constructed a probabilistic proof system with the properties needed to get a lower bound of $n^{1-\epsilon}$.

Since the hardness result holds for any arbitrarily small constant ϵ, the next logical step to improve the lower bound is to show inapproximability results for non-constant ϵ. However, Håstad's proof of the existence of a probabilistic proof system with the needed properties is very long and complicated. This has, until now, hindered any advance in this direction, but recently, Samorodnitsky and

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 2–12, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Trevisan [11] constructed another probabilistic proof system with the needed properties, but where the proof of correctness is much simpler. Armed with this new construction, new results are within reach.

In this paper, we show that it is indeed impossible to approximate Max Clique in polynomial time within $n^{1-\epsilon}$ where $\epsilon \in O(1/\sqrt{\log\log n})$, given that **NP** does not admit randomized algorithms with slightly super-polynomial expected running time. To do this we first ascertain that the reductions from probabilistic proof systems to Max Clique [8,3,12] work also in the case of a non-constant ϵ. This has the additional bonus of collecting in one place the various parts of the reduction, which were previously scattered in the literature. We also extend the previously published reductions to be able to use the construction of Samorodnitsky and Trevisan [11], which characterizes **NP** in terms of a probabilistic proof system with so called non-perfect completeness. To our knowledge, such reductions have not appeared explicitly in the literature before.

When we combine the new reductions with the probabilistic proof system of Samorodnitsky and Trevisan [11], we obtain the following concrete result regarding the approximability of Max Clique:

Theorem 1. *Unless* $\mathbf{NP} \subseteq \mathbf{ZPTIME}(2^{O(\log n(\log\log n)^{3/2})})$, *Max Clique on a graph with n vertices cannot be approximated within $n^{1-O(1/\sqrt{\log\log n})}$ in polynomial time.*

As a comparison, the best known polynomial time approximation algorithm [5], approximates Max Clique within $n^{1-O(\log\log n/\log n)}$. We omit several proofs from this extended abstract. They are contained in the full version of the paper, available from the authors' home page.[1]

2 Preliminaries

Definition 1. *Let P be an* **NP** *maximization problem. For an instance x of P let* $\mathrm{opt}(x)$ *be the optimal value. A solution y with weight $w(x,y)$, is c-*approximate *if it is feasible and $w(x,y) \geq \mathrm{opt}(x)/c$.*

Definition 2. *A c-*approximation algorithm *for an* **NP** *optimization problem P is a polynomial time algorithm that for any instance x and any input y outputs a c-approximate solution.*

We use the wording *to approximate within c* as a synonym for *to compute a c-approximate solution.*

Definition 3. *Max Clique is the following maximization problem: Given a graph $G = (V,E)$ find the largest possible $C \subseteq V$ such that if v_1 and v_2 are vertices in C, then (v_1, v_2) is an edge in E.*

[1] http://www.nada.kth.se/~enge/

Definition 4. *G-gap E3-Sat-5 is the following decision problem: We are given a Boolean formula ϕ in conjunctive normal form, where each clause contains exactly three literals and each literal occurs exactly five times. We know that either ϕ is satisfiable or at most a fraction G of the clauses in ϕ are satisfiable and are supposed to decide if the formula is satisfiable.*

We know from [7] that G-gap E3-Sat-5 is **NP**-hard.

2.1 Previous Hardness Results

A language L is in the class **NP** if there exists a polynomial time Turing machine M, with the following properties:

- For instances $x \in L$, there exists a proof π, of size polynomial in $|x|$, such that M accepts (x, π).
- For instances $x \notin L$, M does not accept (x, π) for any proof π of size polynomial in $|x|$.

Arora and Safra [2] used a generalization of the above definition of **NP** to define the class $\mathbf{PCP}[r, q]$, consisting of a probabilistically checkable proof system (PCP) where the verifier has oracle access to the membership proof, is allowed to use $r(n)$ random bits and query $q(n)$ bits from the oracle.

Definition 5. *A probabilistic polynomial time Turing machine V with oracle access to π is an* (r, q)-restricted verifier *if it, for every oracle π and every input of size n, uses at most $r(n)$ random bits and queries at most $q(n)$ bits from the oracle. We denote by V^π the verifier V with the oracle π fixed.*

Definition 6. *A language L belongs to the class $\mathbf{PCP}[r, q]$ if there exists a (r, q)-restricted verifier V with the following properties:*

- *For instances $x \in L$, $\Pr_\rho[V^\pi$ accepts $(x, \rho)] = 1$ for some oracle π.*
- *For instances $x \notin L$, $\Pr_\rho[V^\pi$ accepts $(x, \rho)] \leq 1/2$ for all oracles π.*

Above, ρ is the random string of length r.

The connection between the approximability of Max Clique and PCPs was first explored by Feige et al. [8], who showed that

$$\mathbf{NP} \subseteq \mathbf{PCP}[O(\log n \log\log n), O(\log n \log\log n)] \tag{1}$$

and used this characterization of **NP** and a reduction to show that Max Clique cannot be approximated within any constant unless

$$\mathbf{NP} \subseteq \mathbf{DTIME}(n^{O(\log\log n)}). \tag{2}$$

The assumption on **NP** needed to prove hardness result on the approximability of Max Clique is closely related to the connection between the classes **NP** and $\mathbf{PCP}[r, q]$ for various values of r and q. This connection was the subject of intensive investigations leading to the following result of Arora et al. [1]:

Theorem 2. $\mathbf{NP} = \mathbf{PCP}[O(\log n), O(1)]$.

A consequence of this result is that the abovementioned assumptions in the proof of Feige et al. [8] could be weakened to $\mathbf{P} = \mathbf{NP}$.

A technical tool in the proof of Feige et al. [8] is the construction of a graph $G_{V,x}$, corresponding to a verifier in some proof system and some input x.

Definition 7. *From a verifier V and some input x, we construct a graph $G_{V,x}$ as follows: Every vertex in $G_{V,x}$ corresponds to an accepting computation of the verifier. Two vertices in $G_{V,x}$ are connected if they correspond to* consistent computations*. Two computations Π_1 and Π_2 are consistent if, whenever some bit b is queried from the oracle, the answers are the same for both Π_1 and Π_2.*

In the original construction, the number of vertices in $G_{V,x}$ was bounded by $2^{r(n)+q(n)}$, where $r(n)$ is the number of random bits used by the verifier and $q(n)$ is the number of bits the verifier queries from the oracle. Feige et al. suggest in their paper that the bound on the number of vertices in $G_{V,x}$ could be improved, and it was later recognized that the number of vertices can be bounded by $2^{r(n)+f(n)}$, where $f(n)$ is the *free bit complexity.*

Definition 8. *A verifier has* free bit complexity f *if the number of accepting computations is at most 2^f for any outcome of the random bits tossed by the verifier.*

Definition 9. *A language L belongs to the class $\mathbf{FPCP}_{c,s}[r, f]$ if there exists verifier V with free bit complexity f that given an input x and oracle access to π tosses r independent random bits ρ and has the following properties:*

- *for instances $x \in L$, $\Pr_\rho[V^\pi$ accepts $(x, \rho)] \geq c$ for some oracle π.*
- *for instances $x \notin L$, $\Pr_\rho[V^\pi$ accepts $(x, \rho)] \leq s$ for all oracles π.*

We say that V has completeness c *and* soundness s.

To understand the intuition behind the free bit complexity of a proof system, it is perhaps best to study the behavior of a typical verifier in a typical proof system. Such a verifier first reads a number of bits, the free bits, from the oracle. From the information obtained from those bits and the random string, the verifier determines a number of bits, the non-free bits, that it should read next from the oracle and the values these bits should have in order for the verifier to accept. Finally, the verifier reads these bits from the oracle and check if they have the expected values.

Theorem 3. *Suppose that $L \in \mathbf{FPCP}_{c,s}[r, f]$. Let x be some instance of L, and construct the graph $G_{V,x}$ as in Definition 7. Then, there is a clique of size at least $c2^r$ in $G_{V,x}$ if $x \in L$, and there is no clique of size greater than $s2^r$ if $x \notin L$.*

Proof. First suppose that $x \in L$. Then there exists an oracle such that a fraction c of all random strings make the verifier accept. The computations corresponding to the same oracle are always consistent, and thus there exists a clique of size at least $c2^r$ in $G_{V,x}$.

Now suppose that $x \notin L$ and that there is a clique of size greater than $s2^r$ in $G_{V,x}$. Since vertices corresponding to the same random string can never represent consistent computations, the vertices in the clique all correspond to different random strings. Thus, we can use the vertices to form an oracle making the verifier accept with probability larger than s. This contradicts the assumption that the PCP has soundness s.

Corollary 1. *Suppose that* $\mathbf{NP} \subseteq \mathbf{FPCP}_{c,s}[O(\log n), f]$ *for some constants* c, s, *and* f. *Then it is impossible to approximate Max Clique within* c/s *in polynomial time unless* $\mathbf{P} = \mathbf{NP}$.

Proof. Let L be some **NP**-complete language and x be some instance of L. Let B be some polynomial time algorithm approximating Max Clique within c/s.

The following algorithm decides L: Construct the graph $G_{V,x}$ corresponding to the instance x. Now run B on $G_{V,x}$. If B determines that $G_{V,x}$ has a clique containing more than $s2^r$ vertices, where $r \in O(\log(n))$ is the number of random bits used by the verifier, accept x, otherwise reject.

Since the number of random bits used by the verifier is logarithmic and the number of free bits is a constant, the graph $G_{V,x}$ has polynomial size. Since B is a polynomial time algorithm, the above algorithm also runs in polynomial time.

It is possible to improve on the above result by *gap amplification*. The simplest form of gap amplification is to simply run a constant number of independent runs of the verifier. If any of the rounds causes the verifier to reject, we reject, otherwise we accept. This shows that, for any constant k,

$$\mathbf{FPCP}_{c,s}[r, f] \subseteq \mathbf{FPCP}_{c^k,s^k}[kr, kf], \tag{3}$$

for any functions c, s, r, and f, which strengthens Corollary 1 to

Corollary 2. *Suppose that* $\mathbf{NP} \subseteq \mathbf{FPCP}_{c,s}[O(\log n), f]$ *for some constants* c, s, *and* f. *Then it is impossible to approximate Max Clique within any constant in polynomial time unless* $\mathbf{P} = \mathbf{NP}$.

The above procedure can improve the inapproximability result from a specific constant c/s to any constant, but to improve the inapproximability result from n^α to $n^{\alpha'}$ for some constants α and α', we have to use a more sophisticated form of gap amplification. Also, the concept of free bit complexity needs to be refined. To see why the above procedure fails in this case, suppose that we have some proof system which gives a graph $G_{V,x}$ with $n = 2^{r+f}$ vertices such that we can deduce that it is impossible to approximate Max Clique within n^α in polynomial time. Put another way, this particular proof system has $c/s = n^\alpha$. Now we try to apply the above gap amplification technique. Then we get a new graph $G_{V',x}$ with $2^{kr+kf} = n^k$ vertices and a new inapproximability factor $c^k/s^k = n^{k\alpha}$. Thus, we have failed to improve the lower bound. Obviously, it is not only the free bit complexity of a proof system that is important when it comes to proving lower bounds for Max Clique, but also the *gap*, the quotient of the soundness

and the completeness. We see above that an exponential increase in the gap does not give us anything if the free bit complexity and the number of random bits increase linearly. Bellare and Sudan [4] recognized that the interesting parameter is $f/\log s^{-1}$ in the case of perfect completeness. This parameter was later named the *amortized free bit complexity* and denoted by $\bar{f}$. Note that the above gap amplification does not change $\bar{f}$. Two methods which do improve the lower bound in the case above by keeping down the number of random bits needed to amplify the gap have appeared in the literature [3,12], and both prove the same result: If every language in **NP** can be decided by a proof system with logarithmic randomness, perfect completeness, and amortized free bit complexity $\bar{f}$, then Max Clique cannot be approximated within $n^{1/(1+\bar{f})-\epsilon}$ in polynomial time, unless **NP** = **ZPP**. The constructions are valid for any constant $\bar{f}$ and some arbitrarily small constant $\epsilon > 0$, and they use the same principle as the above gap amplification: They perform consecutive, although not independent, runs of the verifier and accept if all runs accept.

2.2 A New Amortized Free Bit Complexity

For the case of non-perfect completeness, Bellare et al. [3] define the amortized free bit complexity as $f/\log(c/s)$. In this paper, we propose that this definition should be modified.

Definition 10. *The* amortized free bit complexity *for a PCP with free bit complexity f, completeness c and soundness s is*

$$\bar{f} = \frac{f + \log c^{-1}}{\log(c/s)}. \tag{4}$$

Note that both this definition and the previous one reduce to $f/\log s^{-1}$ in the case of perfect completeness, i.e., when $c = 1$. Note also that the above gap amplification does not change the amortized free bit complexity, neither with the original definition nor with our proposed modification of the definition. However, our proposed definition is robust also with respect to the following: Suppose that we modify the verifier in such a way that it guesses the value of the first free bit. This lowers the free bit complexity by one, and halves the completeness and the soundness of the test. With our proposed definition, the amortized free bit complexity does not change, while it decreases with the definition of Bellare et al. [3]. In the case of perfect completeness, the lower bound on the approximability increases as the amortized free bit complexity decreases. This makes it dubious to have a definition in the general case that allows the free bit complexity to be lowered by a process as the above. Using our proposed definition of the free bit complexity, we first establish that the construction of Zuckerman [12] works also in the case of non-constant parameters:

Theorem 4. *If* $\mathbf{NP} \subseteq \mathbf{FPCP}_{1,s}[r, f]$*, then, for any* $r \in \Omega(\log n)$ *and any* $R > r$*, it is impossible to approximate Max Clique in a graph with* N *vertices within* $N^{1/(1+\bar{f})-r/R}$ *in polynomial time unless*

$$\mathbf{NP} \subseteq \mathbf{coRTIME}\big(2^{\Theta(R+\bar{f}+R\bar{f})}\big). \tag{5}$$

In the case where $\bar{f}$ is some constant and $r \in \mathrm{O}(\log n)$, this reduces to the well known theorem that Max Clique cannot be approximated within $n^{1/(1+\bar{f})-\epsilon}$, for any constant $\epsilon > 0$, unless $\mathbf{NP} = \mathbf{ZPP}$. To see this, just choose $R = r/\epsilon$ above. We also investigate the case of non-perfect completeness. By using the same approach as above—performing consecutive, although not independent, runs of the verifier and accepting if all runs accept—we obtain the following theorem, which is implicit in the works of Bellare et al. [3] and Zuckerman [12]:

Theorem 5. *If* $\mathbf{NP} \subseteq \mathbf{FPCP}_{c,s}[r, f]$*, then, for any* $r \in \Omega(\log n)$*, and any* $R > r$ *such that* $c^D 2^R/2 > 2^r$*, where* $D = (R+2)f/\log s^{-1}$*, Max Clique in a graph with* N *vertices cannot be approximated within* $N^{1/(1+\bar{f})-(r+3)/(R+2)}$ *in polynomial time unless*

$$\mathbf{NP} \subseteq \mathbf{BPTIME}\big(2^{\Theta(R+\bar{f}+R\bar{f})}\big) \tag{6}$$

Note that we in our applications choose R such that the term $(r+1)/R$ in the above theorem is small.

When amplifying the gap of a PCP with non-perfect completeness, it seems more natural to use an accept condition different from the above: Instead of accepting when all runs of the verifier accept, accept when some fraction ν of the runs accept. We investigate the consequences of this new condition and show that using that condition we can construct a reduction without two-sided error also in the case of non-perfect completeness. The parameters of interest turns out to be

$$F_\nu = \frac{f + (1-\nu)\log(q - f + 1) + 1}{-\mathcal{H}(\nu, s)}. \tag{7}$$

where q is the number of query bits in the verifier, ν is a parameter which is arbitrarily close to c, and

$$\mathcal{H}(\nu, s) = -\nu \log \frac{\nu}{s} - (1-\nu)\log\frac{1-\nu}{1-s}. \tag{8}$$

We can then prove the following theorem:

Theorem 6. *Suppose every language in* $\mathbf{NP}$ *can be decided by a PCP with completeness* c*, soundness* s*, query complexity* q*, and free bit complexity* f*. Let* μ *and* ν *be any constants such that* $\mu > 0$ *and* $s < \nu < c$*. Let* $h = ((1+\mu)c - \mu - \nu)/(1-\nu)$*. Then, for any* $R > r$*, it is impossible to approximate Max Clique in a graph with* $N = 2^{R+(R+2)F_\nu}$ *vertices within*

$$N^{1/(1+F_\nu)-r/R-(\log h^{-1})/R} \tag{9}$$

by algorithms with expected polynomial running time unless

$$\mathbf{NP} \subseteq \mathbf{ZPTIME}(2^{\Theta(R+F_\nu+RF_\nu)}). \tag{10}$$

If F_ν is a constant and $r(n) \in \mathrm{O}(\log n)$, the above theorem says that Max Clique is hard to approximate within $N^{1/(1+F_\nu)-\epsilon-\mathrm{o}(1)}$, for ν arbitrarily close to c if we choose $\nu = (1+\mu)c - 2\mu$, μ small enough and $R = r/\epsilon$ and in the above theorem.

This might seem worse than in the case with two-sided error, where the interesting parameter was $\bar{f} = (f + \log c^{-1})/\log(c/s)$ instead of F_ν. When $c = 1/2$, s is small and ν is close to c, this is indeed the case—then F_ν is about $2\bar{f}$. However, when c is close to 1, s is small and the PCP has reasonable low query complexity, we expect $\bar{f}$ and F_ν to be close.

3 Hardness of Approximating Max Clique

In their recent paper, Samorodnitsky and Trevisan [11] give a new PCP for **NP** with optimal amortized query complexity, $1+\epsilon$ for any constant $\epsilon > 0$.

Theorem 7 (Implicit in [11]). *For any positive integer k and any constants $\epsilon > 0$ and $\delta > 0$,*

$$\mathbf{NP} \subseteq \mathbf{FPCP}_{(1-\epsilon)^{k^2}, 2^{-k^2}+\delta}[O(\log n), 2k]. \tag{11}$$

This result implies that the test has free bit complexity ϵ, for any constant $\epsilon > 0$. Since the construction is much simpler than the construction of Håstad [9], with reasonable effort it is possible to work through the construction with a non-constant ϵ. This yields the following theorem (we omit the proof):

Theorem 8. *For any increasing function $k(n)$ and any decreasing functions $\epsilon(n) > 0$ and $\delta(n) > 0$, G-gap* E3-Sat-5 *has a PCP which has query complexity $q = k^2+2k$, free bit complexity $f = 2k$, completeness $c \geq (1-\epsilon)^{k^2}$, soundness $s \leq 2^{-k^2} + \delta$, and uses*

$$r \leq C'_G(\log n + 3k)\log\big((2\epsilon^{-1})\delta^{-2}\big) + \big(2k + k^2\log\epsilon^{-1}\big)\big((2\epsilon^{-1})\delta^{-2}\big)^{C''_G} \tag{12}$$

random bits, for some constants C'_G and C''_G.

When we combine the above theorem with Theorem 6, we obtain the proof of Theorem 1

Proof (of Theorem 1). The proof is just a matter of finding suitable choices for the parameters involved: q, f, s, c, k, and R. By putting $\epsilon = k^{-2}$ and $\delta = 2^{-k^2}$ in Theorem 8, we get that $c \geq e^{-1}$, $s \leq 2^{1-k^2}$, and

$$r \leq C'_G(\log n + 3k)\log(2k^2 2^{2k^2}) + (2k + 2k^2\log k)(2k^2 2^{2k^2})^{C''_G}. \tag{13}$$

If we let

$$k(n) = c_0\sqrt{\log\log n}, \tag{14}$$

where $c_0^2 < 1/2C''_G$, we get

$$k^2 < (\log\log n)/2C''_G, \tag{15}$$

$$2^{2k^2} < (\log n)^{1/C''_G}, \tag{16}$$

which implies that r is dominated by the first term. If we substitute our choice of k in this term, we obtain $r = O(\log n \log\log n)$. Now we set $\nu = (1+\mu)c - 2\mu$, where μ is some arbitrary constant such that $s < \nu < c$. Then

$$-\mathcal{H}(\nu, s) = \nu k^2 + O(1), \tag{17}$$

$$F_\nu = \frac{2k + O(\log k)}{\nu k^2 + O(1)} = \frac{2}{\nu k} + o(1/k). \tag{18}$$

If we set $R = r/F_\nu$ in Theorem 6 we get that

$$h = \frac{(1+\mu)c - \mu - \nu}{1-\nu} > \mu \tag{19}$$

and that it is impossible to approximate Max Clique in a graph with $N = 2^{r/F_\nu + r + 2F_\nu}$ vertices within

$$N^{1/(1+F_\nu) - r/R - (\log h^{-1})/R} \geq N^{1-2F_\nu - o(F_\nu)} \tag{20}$$

in polynomial time, unless

$$\mathbf{NP} \subseteq \mathbf{ZPTIME}(2^{\Theta(r/F + F + r)}) = \mathbf{ZPTIME}(2^{\Theta(\log n (\log\log n)^{3/2})}) \tag{21}$$

Now, we want to express this ratio in terms of N. If we insert Eq. 18 in Eq. 20, we get that

$$N^{1-2F_\nu - o(F_\nu)} = N^{1-4/\nu k + o(1/k)}, \tag{22}$$

and if we insert Eq. 14 into this we get that

$$N^{1-2F_\nu - o(F_\nu)} = N^{1-4/\nu c_0 \sqrt{\log\log n} + o(1/\sqrt{\log\log n})}. \tag{23}$$

Since $\sqrt{\log\log N} = \sqrt{\log\log n}(1 + o(1))$,

$$o\Big(\sqrt{\log\log N}\Big) = o\Big(\sqrt{\log\log n}\Big) \tag{24}$$

and

$$\begin{aligned} \frac{1}{\sqrt{\log\log N}} &= \frac{1}{\sqrt{\log\log n}} \cdot \frac{1}{1+o(1)} \\ &= \frac{1}{\sqrt{\log\log n}}\big(1 - o(1)\big) \\ &= \frac{1}{\sqrt{\log\log n}} - o\Big(\frac{1}{\sqrt{\log\log n}}\Big). \end{aligned} \tag{25}$$

Thus,

$$N^{1-2F_\nu - o(F_\nu)} = N^{1-c_1/\sqrt{\log\log N} - o(1/\sqrt{\log\log N})}, \tag{26}$$

where $c_1 = 4/\nu c_0$.

Note that we do not gain anything if we use Theorem 5 instead of Theorem 6. In the former case we get

$$\bar{f} = \frac{2k + O(1)}{k^2 + O(1)} = \frac{2}{k} + o(1/k). \tag{27}$$

and to get a reasonable value for r, we need to set $k^2 = O(\log \log n)$. Thus we get the same hardness result, except for the constant, but with a stronger assumption—$\mathbf{NP} \not\subseteq \mathbf{BPTIME}(\cdot)$ instead of $\mathbf{NP} \not\subseteq \mathbf{ZPTIME}(\cdot)$—if we use Theorem 5.

4 Future Work

An obvious way to improve this result would be to weaken the assumptions on **NP** we used in our hardness result. Best of all, of course, would be to construct deterministic reductions, since this would allow us to replace the probabilistic complexity classes with deterministic ones in all our assumptions on **NP**. Until this is done, an interesting open question is to determine the best definition of the amortized free bit complexity. We have proposed that the definition should be

$$\bar{f} = \frac{f + \log c^{-1}}{\log(c/s)}. \tag{28}$$

This definition works well in the sense that a PCP with one-sided error gives a hardness result for Max Clique under the assumption that **NP**-complete problems cannot be decided with one-sided error in probabilistic polynomial time, and similarly a PCP with two-sided error gives a hardness result for Max Clique under the assumption that **NP**-complete problems cannot be decided with two-sided error in probabilistic polynomial time.

However, we have seen in Theorem 6 that if one wants to use a PCP with two-sided error to obtain hardness results under the assumption that **NP**-complete problems cannot be decided with one-sided error in probabilistic polynomial time, the interesting parameter is (close to) F_c, defined in Eq. 7. To establish whether it is possible to improve this to our proposed definition of $\bar{f}$, or if F_c is the best possible in this case is an interesting open question.

Trying to obtain an upper bound is also interesting, especially since it is currently unknown how well the Lovász ϑ-function approximates Max Clique. Feige [6] has shown that it cannot approximate Max Clique within $n/2^{c\sqrt{\log n}}$, but, in light of Håstad's results [9] and the results of this paper, this does not compromise the Lovász ϑ-function. It may very well be that it beats the combinatorial algorithm of Boppana and Halldórsson [5].

References

1. S. Arora, C. Lund, R. Motwani, M. Sudhan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, May 1998.

2. S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, Jan. 1998.
3. M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs and non-approximability—towards tight results. *SIAM J. Comput.*, 27(3):804–915, June 1998.
4. M. Bellare and M. Sudan. Improved non-approximability results. In *Proc. Twenty-sixth Ann. ACM Symp. on Theory of Comp.*, pages 184–193, Montréal, Québec, May 1994. ACM Press.
5. R. Boppana and M. M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *Bit*, 32(2):180–196, June 1992.
6. U. Feige. Randomized graph products, chromatic numbers, and the Lovasz ϑ-function. In *Proc. Twenty-seventh Ann. ACM Symp. on Theory of Comp.*, pages 635–640, Las Vegas, Nevada, May 1995. ACM Press.
7. U. Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, July 1998.
8. U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *J. ACM*, 43(2):268–292, Mar. 1996.
9. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proc. 37th Ann. IEEE Symp. on Foundations of Comput. Sci.*, pages 627–636, Burlington, Vermont, Oct. 1996. IEEE Computer Society.
10. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.
11. A. Samorodnitsky and L. Trevisan. Notes on a PCP characterization of NP with optimal amortized query complexity. Manuscript, May 1999.
12. D. Zuckerman. On unapproximable versions of NP-complete problems. *SIAM J. Comput.*, 25(6):1293–1304, Dec. 1996.

Approximating the Independence Number and the Chromatic Number in Expected Polynomial Time

Michael Krivelevich[1] and Van H. Vu[2]

[1] Department of Mathematics, Faculty of Exact Sciences,
Tel Aviv University, Tel Aviv 69978, Israel.
krivelev@math.tau.ac.il
[2] Microsoft Research, 1 Microsoft Way, Redmond, WA 98052, USA.
vanhavu@microsoft.com

Abstract. The independence number of a graph and its chromatic number are hard to approximate. It is known that, unless $coRP = NP$, there is no polynomial time algorithm which approximates any of these quantities within a factor of $n^{1-\epsilon}$ for graphs on n vertices.
We show that the situation is significantly better for the average case. For every edge probability $p = p(n)$ in the range $n^{-1/2+\epsilon} \leq p \leq 3/4$, we present an approximation algorithm for the independence number of graphs on n vertices, whose approximation ratio is $O((np)^{1/2}/\log n)$ and whose expected running time over the probability space $G(n,p)$ is polynomial. An algorithm with similar features is described also for the chromatic number.
A key ingredient in the analysis of both algorithms is a new large deviation inequality for eigenvalues of random matrices, obtained through an application of Talagrand's inequality.

1 Introduction

An *independent set* in a graph $G = (V, E)$ is a subset of vertices spanning no edges. The *independence number* of G, denoted by $\alpha(G)$, is the size of a largest independent set in G. A *coloring* of G is a partition $V = C_1 \ldots C_k$ of its vertex set V, in which every part (color class) C_i forms an independent set. The *chromatic number* $\chi(G)$ of G is the minimal possible number of colors in a coloring of G.

Independence number and chromatic number are essential notions in combinatorics and the problem of estimating these parameters is central in both graph theory and theoretical computer science. Unfortunately, it turns out that both of these problems are notoriously difficult. Computing the exact value of $\alpha(G)$ or $\chi(G)$ is known to be NP-hard since the seminal paper of Karp [16].

Given these hardness results, one still hopes to approximate the above parameters within a reasonable factor in polynomial time. For a number $f \geq 1$, we say that an algorithm A approximates the independence number within factor f over graphs on n vertices, if for every such graph G A outputs an independent set I, whose size satisfies $|I| \geq \alpha(G)/f$. Similarly, A approximates the chromatic

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 13–24, 2000.
© Springer-Verlag Berlin Heidelberg 2000

number within factor f for graphs on n vertices if for every such graph it outputs a coloring with the number of colors k satisfying $k \leq f\chi(G)$. We refer the reader to a survey book [15] for a detailed discussion of approximation algorithms.

However, recent results have shown that in the worst case both computational problems are also hard to approximate. Håstad [14] showed that unless $coPR = NP$, there is no approximation algorithm for the independence number whose approximation ratio over graphs on n vertices is less than $n^{1-\epsilon}$ for any fixed $\epsilon > 0$. Also, Feige and Kilian proved in [9] the same hardness result for the chromatic number. In this paper we aim to show that the situation is significantly better when one considers the average case and not the worst case. When discussing the performance of an algorithm A in the average case, it is usually assumed that a probability distribution on the set of all inputs of A is defined. The most widely used probability measure on the set of all graphs on n vertices is the random graph $G(n,p)$. For an integer n and a function $0 \leq p = p(n) \leq 1$, the *random graph* $G(n,p)$ is a graph on n labeled vertices $1,\ldots,n$, where each pair of vertices (i,j) is chosen to be an edge of G independently and with probability p. We say that a graph property P holds *almost surely*, or a.s. for brevity, in $G(n,p)$ if the probability that a graph G, drawn according to the distribution $G(n,p)$, has P tends to 1 as the number of vertices n tends to infinity.

As with many other graph parameters, it turns out that the average case is much simpler to handle than the worst case for both independence and chromatic numbers. Bollobás [5] and Łuczak [19] showed that a. s. the chromatic number of $G(n,p)$ satisfies $\chi(G) = (1+o(1))n\log_2(1/(1-p))/\log_2 n$ for a constant p, and $\chi(G) = (1+o(1))np/(2\ln(np))$ for $C/n \leq p(n) \leq o(1)$. It follows easily from these results that a.s. $\alpha(G(n,p)) = (1-o(1))\log_2 n/\log_2(1/(1-p))$ for a constant p, and $\alpha(G(n,p) = (1-o(1))2\ln(np)/p$ for $C/n \leq p \leq o(1)$. Also, the greedy algorithm, coloring vertices of G one by one and picking each time the first available color for a current vertex, is known to produce a.s. in $G(n,p)$ with $p \geq n^{\epsilon-1}$ a coloring whose number of colors is larger than the optimal one by only a constant factor (see Ch. 11 of the monograph of Bollobás [4]). Hence the largest color class produced by the greedy algorithm is a.s. smaller than the independence number only by a constant factor.

Note however that the above positive statement about the performance of the greedy algorithm hides in fact one quite significant point. While being very successful in approximating the independence/chromatic number for *most* of the graphs in $G(n,p)$, the greedy algorithm may fail miserably for some "hard" graphs on n vertices. It is quite easy to fool the greedy algorithm by constructing an example of a graph on n vertices, for which the ratio of the number of colors used by the greedy algorithm and the chromatic number will be close to n. Moreover, it has been shown by Kučera [17] that for any fixed $\epsilon > 0$ there exists a graph G on n vertices for which, even after a random permutation of vertices, the greedy algorithm produces a. s. a coloring in at least $n/\log_2 n$ colors, while the chromatic number of G is at most n^{ϵ}. Thus, we cannot say that the greedy algorithm is *always* successful.

In contrast, here our goal is to develop approximation algorithms which will work relatively well for *all* graphs on n vertices and whose *expected* running time will be polynomial in n. Given an algorithm A, whose domain is the set of all graphs on n vertices, and a probability space $G(n,p)$, the *expected running time* of A over $G(n,p)$ is defined as $\sum_G Pr[G]R_A(G)$, where the sum runs over all labeled graphs on n vertices, $Pr[G]$ is the probability of G in $G(n,p)$, and $R_A(G)$ stands for the running time of A on G. Thus, while looking for an algorithm A whose expected running time is polynomial, we can allow A to spend a superpolynomial time on some graphs on n vertices, but it should be effective on average.

The approach of devising deterministic algorithms with expected polynomial time over some probability spaces has been undertaken by quite a few papers. Some of them, e.g., [8], [20], [12] discuss coloring algorithms with expected polynomial time. We wish to stress that the underlying probability spaces in all these papers are different from $G(n,p)$.

In this paper we present approximation algorithms for the independence number and the chromatic number, whose expected running time is polynomial over the probability spaces $G(n,p)$, when the edge probability $p(n)$ is not too low. We have the following results.

Theorem 1. *For any constant $\epsilon > 0$ the following holds. If the edge probability $p(n)$ satisfies $n^{-1/2+\epsilon} \leq p(n) \leq 3/4$, then there exists a deterministic algorithm, approximating the independence number $\alpha(G)$ within a factor $O((np)^{1/2}/\log n)$ and having polynomial expected running time over $G(n,p)$.*

Theorem 2. *For any constant $\epsilon > 0$ the following holds. If the edge probability $p(n)$ satisfies $n^{-1/2+\epsilon} \leq p(n) \leq 3/4$, then there exists a deterministic algorithm, approximating the chromatic number $\chi(G)$ within a factor $O((np)^{1/2}/\log n)$ and having polynomial expected running time over $G(n,p)$.*

Thus, in the most basic case $p = 1/2$ we get approximation algorithms with approximation ratio $O(n^{1/2}/\log n)$ – a considerable improvement over best known algorithms for the worst case [7], [13], whose approximation ratio is only $O(n/polylog(n))$. Note also that the smaller the edge probability $p(n)$, the better the approximation ratio is in both our results.

Before turning to descriptions of our algorithms, we would like to say a few words about combinatorial ideas forming the basis of their analysis. As is typically the case with developing algorithms whose expected running time is polynomial, we will need to distinguish efficiently between "typical" graphs in the probability space $G(n,p)$, for which it is relatively easy to provide a good approximation algorithm, and "non-typical" ones, which are rare but may be hard for approximating a desired quantity. As these rare graphs will have an exponentially small probability in $G(n,p)$, this will allow us to spend an exponential time on each of them. This in turn will enable to approximate the independence/chromatic number within the desired factor even for these graphs.

A separation between typical and non-typical instances will be made based on the first eigenvalue of an auxiliary matrix, to be defined later. Thus we may

say that our algorithms exploit spectral properties of random graphs. Spectral techniques have proven very successful in many combinatorial algorithms. The ability to compute eigenvalues and eigenvectors of a matrix in polynomial time combined with understanding of the information provided by these parameters can constitute a very powerful tool, capable of solving algorithmic problems where all other methods failed. This is especially true for randomly generated graphs, several successful examples of spectral techniques are [6], [2], [3]. A survey [1] discusses several applications of spectral techniques to graph algorithms.

In order to show that bad graphs have an exponentially small probability in $G(n,p)$, we will prove a new large deviation result for eigenvalues of random symmetric matrices. This result, bounding the tails of the distribution of the first eigenvalue of a random symmetric matrix, is proven by applying the inequality of Talagrand [22] and may be of an independent interest.

The rest of the paper is organized as follows. In Section 2 we provide technical tools to be used in the the proof of correctness of our algorithms. In Section 3 we present an algorithm for approximating the independence number. In Section 4 an algorithm for approximating the chromatic number is described. Section 5 is devoted to concluding remarks.

We wish to note that our results are asymptotic in nature. Therefore all usual asymptotic assumptions apply. In particular, we assume n to be large enough whenever needed. We omit routinely all ceiling and floor signs. No serious attempt is made to optimize constants involved.

2 Preliminaries

In this section we prove technical results needed for the analysis of our approximation algorithms, to be proposed in the next two sections. In the first subsection we analyze the performance of the greedy algorithm on random graphs, the second subsection is devoted to bounding the tails of the first eigenvalue of a random matrix.

2.1 Greedy Algorithm on Random Graphs

Given a graph $G = (V, E)$ and some fixed ordering of its vertices, the greedy coloring algorithm proceeds by scanning vertices of G in the given order and assigning the first available color for a current vertex. The greedy algorithm has long been known to be a quite successful algorithm for almost all graphs in $G(n,p)$, if the edge probability p is not too small. For our purposes, we need to prove that it is also extremely robust, i.e., uses an optimal up to a constant factor number of colors with probability extremely close to 1. We will also prove that the largest color class of the output of the greedy algorithm is of order of the independence number with even higher probability. Throughout this section we assume that the $p(n)$ falls in the range of Theorems 1-2, i.e., satisfies $n^{-1/2+\epsilon} \leq p(n) \leq 3/4$ for some positive constant ϵ. We fix the natural order of the n vertices, i.e., $1, 2, \ldots, n$.

Lemma 1. *The probability in $G(n,p)$ that the largest color class, produced by the greedy algorithm, has size less than $\ln n/(2p)$, is less than 2^{-n}.*

Proof. Set $\alpha_0 = \frac{\ln n}{2p}$, $t = \frac{n}{2\alpha_0}$. We call a family $\mathcal{C} = \{C_1, \ldots, C_t\}$ of t subsets of V *bad* if

1. All C_i are pairwise disjoint;
2. For every vertex $v \in V \setminus \bigcup_{i=1}^t C_i$ and for every $1 \le i \le t$, there is an edge of G connecting v and C_i;
3. For every $1 \le i \le t$, $|C_i| < \alpha_0$.

It is easy to see that if $C_1, \ldots, C_t$ are the first t colors produced by the greedy algorithm, then the family $\mathcal{C} = \{C_1, \ldots, C_t\}$ satisfies requirements 1, 2 above. Thus, if the greedy algorithm fails to produce a color class of size at least α_0 while running on a graph G, the first t colors of its output form a bad family in G.

Fix a collection $\mathcal{C} = \{C_1, \ldots, C_t\}$ with all C_i being pairwise disjoint and of size $|C_i| < \alpha_0$. A vertex $v \in V \setminus \bigcup_{i=1}^t C_i$ is *stuck* with respect to $\mathcal{C}$ if it satisfies condition 2 above. The probability that v is stuck is

$$\prod_{i=1}^{t}(1-(1-p)^{|C_i|}) \le \exp\{-\sum_{i=1}^{t}(1-p)^{|C_i|}\} \le e^{-t(1-p)^{\alpha_0}}.$$

As the events that different vertices outside $\mathcal{C}$ are stuck are mutually independent, we get

$$Pr[\mathcal{C} \text{ bad}] \le \exp\{-t(1-p)^{\alpha_0}|V \setminus \bigcup_{i=1}^{t} C_i|\} \le e^{-t(1-p)^{\alpha_0} n/2} = (1+o(1))e^{-tn^{1/2}/2}.$$

Therefore the probability that $G(n,p)$ contains a bad collection is at most

$$\left(\sum_{i=1}^{\alpha_0-1}\binom{n}{i}\right)^t (1+o(1))e^{-tn^{1/2}/2} \le (1+o(1))\binom{n}{\alpha_0}^t e^{-tn^{1/2}/2}$$
$$\le n^n e^{-tn^{1/2}/2} \le n^n e^{-\frac{n^{3/2}p}{2\ln n}} < 2^{-n}. \qquad \square$$

Lemma 2. *The probability in $G(n,p)$ that the greedy algorithm uses at least $4np/\ln n$ colors it at most $2^{-2np/\ln n}$.*

Proof. The proof presented here is essentially identical to the proof of Theorem 11.14 of the monograph of Bollobás [4], where a somewhat stronger result is proven for the case of a constant p.

Set $k_0 = \frac{2np}{\ln n}$,$k_1 = 2k_0 = \frac{4np}{\ln n}$. Denote by A^k the event that at least k colors are used by the greedy algorithm in coloring $G(n,p)$. Let also, for $k \le j \le n$, B_j^k denote the event that vertex j gets color k. As obviously $A^{k+1} = \cup_{j=k+1}^n B_j^{k+1}$, we get for all $k_0 \le k \le k_1$

$$Pr[A^{k+1}|A^k] \le \sum_{j=k+1}^{n} Pr[B_j^{k+1}|A^k].$$

Let us estimate now $Pr[B_j^{k+1}|A^k]$ for $j \geq k+1$. Suppose $C_1, \ldots, C_k$ are the color classes produced by the greedy algorithm before coloring vertex j. Then

$$Pr[B_j^{k+1}|A^k] = \prod_{i=1}^{k}(1-(1-p)^{|C_i|}) \leq (1-(1-p)^{j/k})^k \leq e^{-(1-p)^{j/k}k} \leq e^{-(1-p)^{n/k}k} .$$

Hence $Pr[A^{k+1}|A^k] \leq ne^{-(1-p)^{n/k_0}k_0} = ne^{-(1-p)^{\frac{\ln n}{2p}}k_0} = (1+o(1))ne^{-n^{-1/2}k_0} \leq \frac{1}{2}$. We derive

$$Pr[A^{k_1}] \leq \prod_{k=k_0}^{k_1-1} Pr[A^{k+1}|A^k] \leq \left(\frac{1}{2}\right)^{k_1-k_0} = 2^{-k_0} . \qquad \square$$

2.2 Large Deviation Result

In this subsection we present a new large deviation result, which will be needed in the analysis of the algorithms. This result is also of independent interest. The proof in this subsection will make use of the following powerful result, due to Talagrand [22].

Let $t_1, \ldots, t_m$ be independent random variables and let $\mathcal{S}$ be the product space (with the product measure) generated by $t_1, \ldots, t_m$. Fix a set $\mathcal{B} \subset \mathcal{S}$. For a non-negative number t, define $\mathcal{B}_t$ as follows

$$\mathcal{B}_t = \{x \in \mathcal{S} | \forall \alpha = (\alpha_1, \ldots, \alpha_m), \exists y \in \mathcal{B} \text{ s.t. } \sum_{x_i \neq y_i} |\alpha_i| \leq t(\sum_{i=1}^{n} \alpha_i^2)^{1/2}\}.$$

Then Talagrand's inequality gives:

$$Pr[\overline{\mathcal{B}_t}]Pr[\mathcal{B}] \leq e^{-t^2/4} .$$

Given a graph G on n vertices and a number $0 < p < 1$, we define a matrix $M = M(G,p) = (m_{ij})_{i,j=1}^n$ as follows:

$$m_{ij} = \begin{cases} 1, & \text{if } i,j \text{ are non-adjacent in } G, \\ -q/p, & \text{otherwise} , \end{cases} \tag{1}$$

where $q = 1-p$. Let $\lambda_1(M) \geq \lambda_2(M) \geq \ldots \geq \lambda_n(M)$ denote the eigenvalues of M.

Lemma 3.

$$Pr[\lambda_1(M) \geq 4(n/p)^{1/2}] \leq 2^{-np/8} .$$

Proof. First notice that $M(G,p)$ is a random symmetric matrix which can be generated as follows. Consider $\binom{n}{2}$ random variables m_{ij}, $1 \leq i < j \leq n$, where $m_{ij} = 1$ with probability q, and $-q/p$ with probability p. Set $m_{ji} = m_{ij}$ and

$m_{ii} = 1$. This observation enables us to apply known results on eigenvalues of random matrices. Füredi and Komlós proved implicitly in [11] that

$$E[\lambda_1(M)] = 2\left(\frac{qn}{p}\right)^{1/2}(1+o(1)) \ .$$

Thus we need to bound the probability of deviation of the first eigenvalue from its mean. Denote by m the median of $\lambda_1(M)$ and set $\mathcal{B}$ to be the set of all matrices M with $\lambda_1(M) \leq m$. Clearly, $Pr[\mathcal{B}] = 1/2$. Assume that for a positive t, a matrix M^0 satisfies $\lambda_1(M^0) \geq m + t$. Then by Courant-Fisher's theorem, there is a vector $\mathbf{x} = (x_1, \ldots, x_n) \in R^n$ with norm 1 such that

$$m + t \leq \mathbf{x}^t M^0 \mathbf{x} = \sum_{1\leq i<j\leq n} 2x_i x_j m^0_{ij} + \sum_{i=1}^{n} x_{ii}^2 m^0_{ii} = 1 + \sum_{1\leq i<j\leq n} 2x_i x_j m^0_{ij} \ .$$

On the other hand, for any matrix $M^1 \in \mathcal{B}$ we have

$$m \geq \mathbf{x}^t M^1 \mathbf{x} = \sum_{1\leq i<j\leq n} 2x_i x_j m^1_{ij} + \sum_{i=1}^{n} x_{ii}^2 m^1_{ii} = 1 + \sum_{1\leq i<j\leq n} 2x_i x_j m^1_{ij} \ .$$

It follows that

$$\sum_{1\leq i<j\leq n} 2x_i x_j (m^0_{ij} - m^1_{ij}) \geq t \ .$$

Set $\alpha_{ij} = 2x_i x_j$ for $1 \leq i < j \leq n$. Since $\mathbf{x}$ has norm 1, we get $\sum_{1\leq i<j\leq n} \alpha_{ij}^2 \leq 2(\sum_{i=1}^n x_i^2)^2 = 2$. Moreover, since $|m^0_{ij} - m^1_{ij}| \leq 1 + q/p = 1/p$,

$$\sum_{ij, m^0_{ij} \neq m^1_{ij}} |\alpha_{ij}| \geq tp \geq \frac{tp}{\sqrt{2}}(\sum_{1\leq i\leq j\leq n} \alpha_{ij}^2)^{1/2} \ .$$

This implies that $M^0 \in \overline{\mathcal{B}_{tp/\sqrt{2}}}$. By Talagrand's inequality

$$Pr[\lambda_1(M) \geq m + t] \leq Pr[\overline{\mathcal{B}_{tp/\sqrt{2}}}] \leq \frac{1}{Pr[\mathcal{B}]e^{(tp)^2/8}} \ .$$

Given that $Pr[\mathcal{B}] = 1/2$, it follows that

$$Pr[\lambda_1(M) \geq m + t] \leq 2e^{-(tp)^2/8}.$$

Now set $\mathcal{B} = \{M | \lambda_1(M) \leq m - t\}$. By a similar argument, we can show that if $\lambda_1(M^0) \geq m$, then $M^0 \in \overline{\mathcal{B}_{tp/\sqrt{2}}}$. This, again by Talagrand's inequality, yields

$$Pr[\lambda_1(M) \leq m - t] \leq 2e^{-(tp)^2/8}.$$

Together, we have

$$Pr[|\lambda_1(M) - m| \geq t] \leq 4e^{-(tp)^2/8} \ ,$$

or equivalently

$$Pr[|\lambda_1(M) - m| \geq \sqrt{8}t/p] \leq 4e^{-t^2} \ ,$$

The problem here is that the median m is not exactly the mean. However, from what we have already proved, it is simple to show that they differ only by $O(1/p)$. Indeed, let $X = \lambda_1(M)$, $Y = pX$, let also $m_1 = pm$ be the median of Y. Then we have $Pr[|m_1) - Y| \geq t] \leq 4e^{-t^2/8}$. Therefore,

$$|m_1 - E[Y]| \leq E[|m_1 - Y|] \leq \int_0^\infty tPr[|Y - m_1| \geq t]dt \leq \int_0^\infty 4te^{-t^2/8}dt = 16 \ ,$$

implying $|E[X] - m| \leq 16/p$. Recalling our assumption about $p(n)$, the claim of the lemma follows directly. □

What is the connection between $\lambda_1(M(G))$ and $\alpha(G)$? The answer is given by the following simple lemma.

Lemma 4. *Let $M = M(G,p)$ be as defined in (1). Then $\lambda_1(M) \geq \alpha(G)$.*

Proof. Let $k = \alpha(G)$. Then M contains a k by k block of all 1's, indexed by the vertices of an independent set of size k. It follows from interlacing that $\lambda_1(M) \geq \lambda_1(1_{k\times k}) = k$. □

The reader has possibly noticed that $\lambda_1(M(G))$ is an upper bound not only for the independence number of G, but for its Lovász Theta-function ([18]). Therefore our Lemma 3 provides in fact a large deviation result also for the Theta-function. We get the following bound:

Lemma 5. *Let $p = p(n)$ satisfy $p(n) = \omega(1)/n$. Then in $G(n,p)$*

$$Pr[\alpha(G) \geq 4(n/p)^{1/2}] \leq Pr[\theta(G) \leq 4(n/p)^{1/2}] \leq 2^{-np/8} \ .$$

3 Approximating the Independence Number

We are now in position to present both of our approximation algorithms. In this section we describe an algorithm for approximating the independence number, while an algorithm for the chromatic number is described in the next section. We assume that the algorithm is given a graph G on n vertices and the value of the edge probability $p(n)$. For a subset $W \subset V$ we denote $\overline{N}(W) = \{v \in V \setminus W : \forall w \in W, (v,w) \notin E(G)\}$.

Step 1. Run the greedy algorithm on G. Let I be a largest color class produced by the greedy algorithm. If $|I| < \ln n/(2p)$, goto *Step 5*;
Step 2. Define matrix $M = M(G,p)$ as given by (1). Compute $\lambda_1(M)$. If $\lambda_1(M) \leq 4(n/p)^{1/2}$, output I;
Step 3. For each $W \subset V$ of size $|W| = \ln n/p$, compute $|\overline{N}(W)|$. If for no W, $|\overline{N}(W)| > (n/p)^{1/2}$, output I;
Step 4. Check all subsets of V of size $(2n/p)^{1/2}$. If none of them is independent, output I;
Step 5. Check all subsets of V and set I to be an independent set of the largest size. Output I.

Let us first check that the output always approximates $\alpha(G)$ within a factor of $O((np)^{1/2}/\log n)$. If an independent set is output at Step 2, its size satisfies $|I| \geq \ln n/(2p)$, and due to Lemma 4 we know that $\alpha(G) \leq \lambda_1(M) \leq 4(n/p)^{1/2}$. Hence the approximation ratio for this case is $O((n/p)^{1/2}/(\ln n/p)) = O((np)^{1/2}/\ln n)$. If I is output at Step 3, then G does not contain an independent set of size $(n/p)^{1/2} + \ln n/p \leq 2(n/p)^{1/2}$, since no W has many non-neighbors. If I is output at Step 4, we get that $\alpha(G) \leq (2n/p)^{1/2}$, thus giving the desired approximation ratio. Finally, if the output is produced at Step 5, it is the result of the exhaustive search over all subsets of V, and thus its size is equal to $\alpha(G)$.

Now it remains to prove that the expected running time of the algorithm is polynomial. We assume that the exhaustive search has running time 2^n (in fact better exponential bound is known, but we do not need it). Notice that eigenvalues of an n by n matrix are computable in time polynomial in n. Clearly, the cost of performing Steps 1 and 2 of the above algorithm is polynomial. The only chance to get to Step 3 is to have a graph G with $\lambda_1(M(G,p)) > 4(n/p)^{1/2}$. The probability of this event is at most $2^{-np/8}$ by Lemma 3. The complexity of Step 3 is $O(\binom{n}{\ln n/p})$. Therefore the expected amount of calculations performed at Step 3 is of order at most

$$\binom{n}{\ln n/p} 2^{-np/8} \leq \left(\frac{enp}{\ln n}\right)^{\ln n/p} 2^{-np/8} = o(1)\,,$$

due to our assumption on $p(n)$. Similarly, we get to Step 4 only if there exists a set W of size $|W| = \ln n/p$ with $\overline{N}(W) \geq (n/p)^{1/2}$. The probability of this event in $G(n,p)$ is at most

$$\binom{n}{\ln n/p}\binom{n}{(n/p)^{1/2}}(1-p)^{(\ln n/p)(n/p)^{1/2}} = o(\binom{n}{(n/p)^{1/2}}^{-1})\ .$$

As executing Step 4 requires $\binom{n}{(n/p)^{1/2}}$ operations, the expected number of operations performed at Step 4 is $o(1)$. Finally, we get to Step 5 if either the greedy algorithm outputs no color class of size at least $\ln n/(2p)$ (and the probability of this event is at most 2^{-n} by Lemma 1) or if G contains an independent set of size $(n/p)^{1/2}$, this happens with probability at most

$$\binom{n}{(2n/p)^{1/2}}(1-p)^{\binom{(2n/p)^{1/2}}{2}} = o(2^{-n})\ .$$

Thus the expected complexity of Step 5 is also $o(1)$. We can conclude that the expected running time of the above algorithm over $G(n,p)$ is dominated by the cost of performing its first two steps (in fact, Step 2) and is therefore polynomial in n. This proves Theorem 1.

4 Approximating the Chromatic Number

In this section we present an approximation algorithm for the chromatic number. We assume again that the algorithm is given a graph G on n vertices and the value of p as an input.

Step 1. Run the greedy algorithm on G. Let $\mathcal{C}_1$ be the resulting coloring. If the number of colors in $\mathcal{C}_1$ is at least $4np/\ln n$ goto *Step 3*;
Step 2. Define $M = M(G,p)$ according to (1) and compute $\lambda_1(M)$. If $\lambda_1(M) \leq 4(n/p)^{1/2}$, output $\mathcal{C}_1$;
Step 3. If the number of vertices of G of degree at least $4np$ exceeds np goto *Step 7*. Otherwise, color G by first coloring each vertex of degree at least $4np$ by a separate color, and then running the greedy algorithm on the rest of the graph and using fresh colors. Let $\mathcal{C}_2$ denote the obtained coloring.
Step 4. For each $W \subset V$ of size $|W| = \ln n/p$, compute $|\overline{N}(W)|$. If for no W, $|\overline{N}(W)| \geq n^{1/2}/(p^{1/2}\ln n)$ output $\mathcal{C}_2$;
Step 5. Check all subsets of V of size $n^{1/2}/(4p^{1/2}\ln n)$. If none of them is independent, output $\mathcal{C}_2$;
Step 6. Check whether there exist $\ln^4 n$ pairwise disjoint independent sets of size $n^{1/2}/(p^{1/2}\ln n)$ each. If there is no such collection output $\mathcal{C}_2$;
Step 7. Find an optimal coloring by the exhaustive search and output it.

As the reader has possibly noticed, the above algorithm is quite similar to that of Section 3. The algorithm of this section is however somewhat more complicated. This distinction is caused by the fact that the bound of Lemma 1 is much stronger than that of Lemma 2.

Let us verify that the above algorithm approximates $\chi(G)$ within a factor of $O((np)^{1/2}/\ln n)$. If coloring $\mathcal{C}_1$ is output at Step 2, we get by Lemma 4 $\chi(G) \geq n/\alpha(G) \geq n/\lambda_1(M) \geq (np)^{1/2}/4$. On the other hand, $\mathcal{C}_1$ has at most $4np/\ln n$ colors. Thus in this case the approximation ratio is $O((np)^{1/2}/\ln n)$. Observe that if G has at most np vertices of degree at least $4np$, then the coloring $\mathcal{C}_2$, produced at Step 3, uses at most $np + 4np = 5np$ colors. If $\mathcal{C}_2$ is output at Step 4, then $\alpha(G) = O(n^{1/2}/(p^{1/2}\ln n))$ and hence the approximation ratio in this case is $O(np/(n^{1/2}p^{1/2}\ln n)) = O((np)^{1/2}/\ln n)$ as well. An identical argument works for Step 5. If $\mathcal{C}_2$ is output at Step 6, we claim that $\chi(G) = \Omega((np)^{1/2}\ln n)$. Indeed, let $V = (C_1, \ldots, C_k)$ be an optimal coloring of G. Let k_0 be the number of color classes of size at least $n^{1/2}/(p^{1/2}\ln n)$, then $k_0 \leq \ln^4 n$. Also, all color classes are of size less than $n^{1/2}\ln^3 n/p^{1/2}$. Thus the k_0 large color classes cover altogether at most $n^{1/2}\ln^7 n/p^{1/2} \ll n$ vertices. The rest of the vertices are covered by $k - k_0$ color classes, each of size less than $n^{1/2}/(p^{1/2}\ln n)$, implying $k \geq k - k_0 \geq (1-o(1))n/(n^{1/2}/(p^{1/2}\ln n)) = (1-o(1))(np)^{1/2}\ln n$. Recalling that $\mathcal{C}_2$ has $O(np)$ colors, we get the desired approximation ratio. Finally, if we ever get to Step 7, the output is found by the exhaustive search and is thus optimal.

The expected running time of the above algorithm can be shown to be polynomial in n similarly to the algorithm for the independence number. The only notable difference is that here we use Lemma 2. We omit detailed calculations.

5 Concluding Remarks

In this paper we presented approximation algorithms for the independence number and the chromatic number of a graph. These algorithms were designed as

to be efficient over the probability space $G(n,p)$ of random graphs for various values of $p = p(n)$. For every $p(n)$ in the range $n^{-1/2+\epsilon} \leq p(n) \leq 0.75$, both our algorithms *always* achieve approximation ratio $O((np)^{1/2}/log(n))$ and their *average* running time is polynomial over $G(n,p)$.

How good are our results? As stated in the introduction, their approximation ratio is much better than that of the best known algorithms for the worst case. Still, the greedy algorithm, one of the simplest possible algorithms for finding an independent set or a coloring, performs much better for a typical graph than what is guaranteed by our approximation algorithms. There is some indication, however, that the approximation ratio $O((np)^{1/2}/\log n)$ may be hard to improve. Consider, for example, the basic case $p = 1/2$. Saks [21] suggested the following interesting problem. Suppose G is a graph on n vertices which has been generated either according to the distribution $G(n,1/2)$ or according to the following distribution: choose first a random graph $G(n,1/2)$ and then pick randomly a subset Q of size k and force it to be independent by erasing all edges inside Q. We denote the last model of random graphs by $G(n,1/2,k)$. The problem is to distinguish in polynomial time between the above two models. For the case $k = \Theta(n^{1/2})$, Alon, Krivelevich and Sudakov [3] showed how to recover the independent set of size k in $G(n,1/2,k)$, using spectral techniques, thus clearly providing a tool for distinguishing between $G(n,1/2)$ and $G(n,1/2,k)$. (See also [10] for a related result.) However, Saks' question is still open for every $k = o(n^{1/2})$. Returning to our problem of developing efficient approximation algorithms, note that if we are unable to distinguish between $G(n,1/2)$ and $G(n,1/2,k)$ in polynomial time when $k = o(n^{1/2})$, our algorithm should act the same for both models. As the independence number is a.s. of order $\ln n$ in the first model and is a.s. k in the second one, we cannot hope then to get an algorithm with approximation ratio better than $k/\ln n$. This argument shows that the question of Saks and the problem of developing good approximation algorithms for the independence/chromatic number may be tightly connected.

An obvious open question is what can be done for smaller values of p, i.e., for $p \ll n^{-1/2}$. While our large deviation result (Lemma 3) keeps working for smaller values of p as well, Step 3 of our algorithm for approximating the independence number does not have polynomial expected time anymore (as $\binom{n}{\ln n/p}2^{-np/8}$ tends exponentially fast to infinity for $p \ll n^{-1/2}$). Using again the greedy algorithm as our main tool, we can give an algorithm whose approximation ratio is $O(np)$. We conjecture however that much better approximation algorithms exist for sparse random graphs.

Spectral techniques combined with large deviation inequalities have played an essential role in both of our algorithms. It appears that this machinery can be used successfully to develop approximation algorithms with expected polynomial time for other hard combinatorial problems as well. One possible candidate is the problem of approximating the value of a maximum cut in a graph (MAXCUT). We hope that the ideas of this paper can be used to devise algorithms, finding almost optimal cuts in expected polynomial time for various values of the edge probability p.

References

1. N. Alon, *Spectral techniques in graph algorithms*, Lecture Notes Comp. Sci. 1380 (C. L. Lucchessi and A. V. Moura, Eds.), Springer, Berlin, 1998, 206–215.
2. N. Alon and N. Kahale, *A spectral technique for coloring random 3-colorable graphs*, Proc. 26th ACM STOC, ACM Press (1994), 346-355.
3. N. Alon, M. Krivelevich and B. Sudakov, *Finding a large hidden clique in a random graph*, Proc. 9th ACM-SIAM SODA, ACM Press (1998), 594–598.
4. B. Bollobás, **Random graphs**, Academic Press, New York, 1985.
5. B. Bollobás, *The chromatic number of random graphs*, Combinatorica 8 (1988), 49–55.
6. R. Boppana, *Eigenvalues and graph bisection: An average case analysis*, Proc. 28th IEEE FOCS, IEEE (1987), 280–285.
7. R. Boppana and M. M. Halldórsson, *Approximating maximum independent sets by excluding subgraphs*, Bit 32 (1992), 180–196.
8. M. Dyer and A. Frieze, *The solution of some random NP-hard problems in polynomial expected time*, J. Algorithms 10 (1989), 451–489.
9. U. Feige and J. Kilian, *Zero knowledge and the chromatic number*, Proc. 11th IEEE Conf. Comput. Complexity, IEEE (1996), 278–287.
10. U. Feige and R. Krauthgamer, *Finding and certifying a large hidden clique in a semi-random graph*, Random Str. Algor. 16 (2000), 195–208.
11. Z. Füredi and J. Komlós, *The eigenvalues of random symmetric matrices*, Combinatorica 1 (1981), 233–241.
12. M. Fürer, C. R. Subramanian and C. E. Veni Madhavan, *Coloring random graphs in polynomial expected time*, Algorithms and Comput. (Hong Kong 1993), Lecture Notes Comp. Sci. 762, Springer, Berlin, 1993, 31–37.
13. M. M. Halldórsson, *A still better performance guarantee for approximate graph coloring*, Inform. Process. Lett. 45 (1993), 19–23.
14. J. Håstad, *Clique is hard to approximate within* $n^{1-\epsilon}$, Proc. 37th IEEE FOCS, IEEE (1996), 627–636.
15. **Approximation algorithms for NP-hard problems**, D. Hochbaum, Ed., PWS Publish. Company, Boston, 1997.
16. R. Karp, Reducibility among combinatorial problems, in: *Complexity of computer computations* (E. Miller and J. W. Thatcher, eds.) Plenum Press, New York, 1972, 85–103.
17. L. Kučera, *The greedy coloring is a bad probabilistic algorithm*, J. Algorithms 12 (1991), 674–684.
18. L. Lovász, *On the Shannon capacity of a graph*, IEEE Trans. Inform. Theory 25 (1979), 1–7.
19. T. Łuczak, *The chromatic number of random graphs*, Combinatorica 11 (1991), 45–54.
20. H. J. Prömel and A. Steger, *Coloring clique-free graphs in polynomial expected time*, Random Str. Algor. 3 (1992), 275–402.
21. M. Saks, Private communication.
22. M. Talagrand, *A new isoperimetric inequality for product measure, and the tails of sums of independent random variables*, Geom. Funct. Analysis 1 (1991), 211–223.

Closed Types as a Simple Approach to Safe Imperative Multi-stage Programming

Cristiano Calcagno[1], Eugenio Moggi[1]*, and Walid Taha[2]**

[1] DISI, Univ. di Genova, Genova, Italy
{calcagno,moggi}@disi.unige.it
[2] Department of Computing Sciences, Chalmers, Göteborg, Sweden
taha@cs.chalmers.se

Abstract. Safely adding computational effects to a multi-stage language has been an open problem. In previous work, a *closed type constructor* was used to provide a safe mechanism for executing dynamically generated code. This paper proposes a general notion of *closed type* as a simple approach to safely introducing computational effects into multi-stage languages. We demonstrate this approach formally in a core language called $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$. This core language combines safely multi-stage constructs and ML-style references. In addition to incorporating state, $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$ also embodies a number of technical improvements over previously proposed core languages for multi-stage programming.

1 Introduction

Many important software applications require the manipulation of open code at run-time. Examples of such applications include high-level program generation, compilation, and partial evaluation [JGS93]. But having a notion of values that includes open code (that is, possibly containing free variables) complicates both the (untyped) operational semantics and type systems for programming languages designed to support such applications. This paper advocates a simple and direct approach for safely adding computational effects into languages that manipulate open code. The approach capitalises on a single type constructor that guarantees that a given term will evaluate to a *closed* value at run-time. We demonstrate our approach in the case of ML-style references [MTHM97].

We extend recent studies into the semantics and type systems for multi-level and multi-stage languages. **Multi-level** languages [GJ91,GJ96,Mog98,Dav96] provide a mechanism for constructing and combining open code. **Multi-stage** languages [TS97,TBS98,MTBS99,BMTS99,Tah99,Tah00] extend multi-level languages with a construct for executing the code generated at run-time. Multi-stage programming can be illustrated using **MetaML** [TS97,Met00], an extension of SML [MTHM97] with a type constructor $\langle_\rangle$ for open code. MetaML provides

* Research partially supported by MURST and ESPRIT WG APPSEM.
** Postdoctoral Fellow funded by the Swedish Research Council for Engineering Sciences (TFR), grant number 221-96-403.

U. Montanari et al.(Eds.): ICALP 2000, LNCS 1853, pp. 25–36, 2000.
© Springer-Verlag Berlin Heidelberg 2000

```
-| datatype nat = z | s of nat;                        (* natural numbers*)
datatype  nat

-| fun p z     x y = (y := 1.0)                  (* conventional program *)
     | p (s n) x y = (p n x y; y := x * !y);
val p = fn : nat -> real -> real ref -> unit

-| fun p_a z     x y = <~y := 1.0>                  (* annotated program *)
     | p_a (s n) x y = <~(p_a n x y); ~y:=~x * !~y>;
val p_a = fn : nat -> <real> -> <real ref> -> <unit>

-| val p_cg =                                        (* code generator *)
      fn n => <fn x y => ~(p_a n <x> <y>)>;
val p_cg = fn : nat  -> <real -> real ref -> unit>

-| val p_sc = p_cg 3;                                (* specialised code *)
val p_sc = <fn x y => (y:=1.0; y:=x*!y; y:=x*!y; y:=x*!y)>
         : <real -> real ref -> unit>

-| val p_sp = run p_sc;                            (* specialised program *)
val p_sp = fn : real -> real ref -> unit
```

Fig. 1. Example of multi-stage programming with references in MetaML

three basic staging constructs that operate on this type: Brackets $\langle _ \rangle$, Escape ~ _ and Run run _. Brackets defers the computation of its argument; Escape splices its argument into the body of surrounding Brackets; and Run executes its argument.

Figure 1 lists a sequence of declarations illustrating the multi-stage programming method [TS97,BMTS99] in an imperative setting:

- `p` is a conventional "single-stage" program, which takes a natural `n`, a real `x`, a reference `y`, and stores x^n in `y`.
- `p_a` is a "two-stage" *annotated* version of `p`, which requires the natural `n` (as before), but uses only symbolic representations for the real `x` and the reference `y`. `p_a` builds a representation of the desired computation. When the first argument is zero, no assignment is performed, instead a piece of code for performing an assignment at a later time is generated. When the first argument is greater than zero, code is generated for performing an assignment at a later time, and moreover the recursive call to `p_a` is performed so that the whole code-generation is performed in full.
- `p_cg` is the *code generator*. Given a natural number, the code generator proceeds by building a piece of code that contains a lambda abstraction, and then using Escape performs an unfolding of the annotated program `p_a` over the "dummy variables" `<x>` and `<y>`. This powerful capability of "evaluation under lambda" is an essential feature of multi-stage programming languages.

- `p_sc` is the *specialised code* generated by applying `p_cg` to a particular natural number (in this case 3). The generated (high-level) code corresponds closely to machine code, and should compile into a light-weight subroutine.
- `p_sp` is the *specialised program*, the ultimate goal of run-time code generation. The function `p_sp` is a specialised version of `p` applied to 3, which does not have unnecessary run-time overheads.

Problem Safely adding computational effects to multi-stage languages has been an open problem[1]. For example, when adding ML-style references to a multi-stage language like MetaML, one can have that "dynamically bound" variables go out of the scope of their binder [TS00]. Consider the following MetaML[2] session:

```
-| val a = ref <1>;
val a = ... : ref <int>
-| val b = <fn x => ~(a:=<x>; <2>)>;
val b = <fn x => 2> : <int -> int>
-| val c = !a;
val c = <x> : <int>
```

In evaluating the second declaration, the variable `x` goes outside the scope of the binding lambda, and the result of the third line is wrong, since x is not bound in the environment, even though the session is well-typed according to naive extensions of previously proposed type systems for MetaML. This form of **scope extrusion** is specific to multi-level and multi-stage languages, and it does *not* arise in traditional programming languages, where evaluation is generally restricted to closed terms (e.g. see [Plo75] and many subsequent studies.) The the problem lies in the *run-time interaction between free variables and references.*

Remark 1. In the type system we propose (see Figure 2) the above session is not well-typed. First, `ref <1>` cannot be typed, because `<1>` is not of a closed type. Second, if we add some closedness annotation to make the first line well-typed, i.e. `val a = ref [<1>]`, then the type of `a` becomes `ref [<int>]`, and we can no longer type `a:=<x>` in the third line. Now, there is no way to add closedness annotations, e.g. `a:=[<x>]`, to make the third line well-typed, in fact the (close)-rule is not applicable to derive $a\colon \mathsf{ref}\ \mathsf{nat}^0; x\colon \mathsf{nat}^1 \vdash [\langle x\rangle]\colon [\langle \mathsf{nat}\rangle]^0$.

Contributions and organisation of this paper This paper shows that *multi-stage and imperative features can be combined safely in the same programming*

[1] The current release of MetaML [Met00] is a substantial language, supporting most features of SML and a host of novel meta-programming constructs. In this release, safety is not guaranteed for meta-programs that use Run or effects. We hope to incorporate the ideas presented in this paper into the next MetaML release.

[2] The observation made here also applies to $\lambda^{\bigcirc}$ [Dav96].

language. We demonstrate this formally using a core language, that we call $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$, which extends Mini-ML [CDDK86] with ML-style references and[3]

- *A code type constructor* $\langle _ \rangle$ [TS97,TBS98,Dav96].
- *A closed type constructor* $[_]$ [BMTS99], but with improved syntax borrowed from $\lambda^{\Box}$ [DP96].
- *A term construct* $\mathsf{run}\ _$ [TS97] typed with $[_]$.

The key technical result is **type safety** for $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$, i.e. evaluation of well-typed programs does not raise an error (see Theorem 1). The type system of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$ is simpler than some related systems for binding-time analysis (BTA), and it is also more expressive than most proposals for such systems (Section 3).

In principle the additional features of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$ should not prevent us from writing programs like those in normal imperative languages. This can be demonstrated by giving an embedding of $\mathsf{Mini\text{-}ML}_{\mathsf{ref}}$ into our language, omitted for brevity. We expect the simple approach of using closed types to work in relation to other computational effects, for example: only closed values can be packaged with exceptions, only closed values can be communicated between processes.

Note on Previous Work The results presented here are a significant generalisation of a recently proposed solution to the problem of assigning a sound type to Run. The naive typing $\mathsf{run} : \langle t \rangle \to t$ of Run is unsound (see [TBS98]), since it allows to execute an arbitrary piece of code, including "dummy variables" such as `<x>`. The **closed type constructor** $[_]$ proposed in [BMTS99] allows to give a sound typing $\mathsf{run} : [\langle t \rangle] \to t$ for Run, since one can *guarantee* that values of type $[\tau]$ will be *closed*. In this paper, we generalise this property of the closed type constructor to a bigger set of types, that we call **closed types**, and we also exploit these types to avoid the scope extrusion problem in the setting of imperative multi-stage programming.

2 $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$

This section describes the syntax, type system and operational semantics of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$, and establishes safety of well-typed programs. The types τ and closed types σ are defined as

$$\tau \in \mathsf{T} ::= \sigma \mid \tau_1 \to \tau_2 \mid \langle \tau \rangle \qquad \sigma \in \mathsf{C} ::= \mathsf{nat} \mid [\tau] \mid \mathsf{ref}\ \sigma$$

Intuitively, a term can only be assigned a closed type σ when it will evaluate to a closed value (see Lemma 4). Values of type $[\tau]$ are always closed, but relying only on the close type constructor makes programming verbose [MTBS99,BMTS99,Tah00]. The generalised notion of closed type greatly improves the usability of the language (see Section 2.3). The set of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$

[3] $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$ can incorporate also MetaML's *cross-stage persistence* [TS97]. This can be done by adding an up, similar to that of λ^{BN} [MTBS99], and by introducing a demotion operation. This development is omitted for space reasons.

terms is parametric in an infinite set of variables $x \in \mathsf{X}$ and an infinite set of locations $l \in \mathsf{L}$

$$\begin{aligned} e \in \mathsf{E} ::= {} & x \mid \lambda x.e \mid e_1\ e_2 \mid \mathsf{fix}\ x.e \mid \mathsf{z} \mid \mathsf{s}\ e \mid (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{z} \to e_1 \mid \mathsf{s}\ x \to e_2) \mid \\ & \langle e \rangle \mid \tilde{}e \mid \mathsf{run}\ e \mid [e] \mid (\mathsf{let}\ [x] = e_1\ \mathsf{in}\ e_2) \mid \\ & \mathsf{ref}\ e \mid !\ e \mid e_1 := e_2 \mid l \mid \mathsf{fault} \end{aligned}$$

The first line lists the Mini-ML terms: variables, abstraction, application, fixpoint for recursive definitions, zero, successor, and case-analysis on natural numbers. The second line lists the three multi-stage constructs of MetaML [TS97]: *Brackets* $\langle e \rangle$ and *Escape* $\tilde{}e$ are for building and splicing code, and *Run* is for executing code. The second line also lists the two "closedness annotations": *Close* $[e]$ is for marking a term as being closed, and *Let-Close* is for forgetting these markings. The third line lists the three SML operations on references, constants l for locations, and a constant fault for a program that crashes. The constants l and fault are not allowed in user-defined programs, but they are instrumental to the operational semantics of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$.

Remark 2. Realistic implementations should erase closedness annotations, by mapping $[e]$ to e and $(\mathsf{let}\ [x] = e_1\ \mathsf{in}\ e_2)$ to $(\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2)$.

The constant fault is used in the rules for symbolic evaluation of binders, e.g. we write $\dfrac{\mu, e \overset{n+1}{\hookrightarrow} \mu', v}{\mu, \lambda x.e \overset{n+1}{\hookrightarrow} \mu'[x := \mathsf{fault}], \lambda x.v}$ instead of $\dfrac{\mu, e \overset{n+1}{\hookrightarrow} \mu', v}{\mu, \lambda x.e \overset{n+1}{\hookrightarrow} \mu', \lambda x.v}$. This more *hygienic* handling of scope extrusion is compatible with the identification of terms modulo α-conversion, and prevents new free variable to appear as effect of the evaluation (see Lemma 3). On the other hand, in implementations there is no need to use the more hygienic rules, because during evaluation of a well-typed program (starting from the empty store) only closed values get stored.

Note 1. We will use the following notation and terminology

- Term equivalence, written $\equiv$, is α-conversion. Substitution of e for x in e' (modulo $\equiv$) is written $e'[x := e]$.
- m, n range over the set N of natural numbers. Furthermore, $m \in \mathsf{N}$ is identified with the set $\{i \in \mathsf{N} | i < m\}$ of its predecessors.
- $f : A \overset{fin}{\to} B$ means that f is a partial function from A to B with a finite domain, written $dom(f)$.
- $\Sigma : \mathsf{L} \overset{fin}{\to} \mathsf{T}$ is a *signature* (for locations only), written $\{l_i : \mathsf{ref}\ \sigma_i | i \in m\}$.
- $\Delta, \Gamma : \mathsf{X} \overset{fin}{\to} (\mathsf{T} \times \mathsf{N})$ are type-and-level assignments, written $\{x_i : \tau_i^{n_i} | i \in m\}$. We use the following operations on type-and-level assignments:
  $\{x_i : \tau_i^{n_i} | i \in m\}^{+n} \overset{\Delta}{=} \{x_i : \tau_i^{n_i+n} | i \in m\}$ adds n to the level of the x_i;
  $\{x_i : \tau_i^{n_i} | i \in m\}^{\leq n} \overset{\Delta}{=} \{x_i : \tau_i^{n_i} | n_i \leq n \wedge i \in m\}$ removes the x_i with level $> n$.
- $\mu : \mathsf{L} \overset{fin}{\to} \mathsf{E}$ is a *store*.
- $\Sigma, l : \mathsf{ref}\ \sigma$, $\Gamma, x : \tau^n$ and $\mu\{l = e\}$ denote extension of a signature, assignment and store respectively.

$$\frac{}{\Sigma, \Delta; \Gamma \vdash x\colon \tau^n}\ \Delta(x) = \tau^n \qquad \frac{}{\Sigma, \Delta; \Gamma \vdash x\colon \tau^n}\ \Gamma(x) = \tau^n$$

$$\frac{\Sigma, \Delta; \Gamma, x\colon \tau_1^n \vdash e\colon \tau_2{}^n}{\Sigma, \Delta; \Gamma \vdash \lambda x.e\colon \tau_1 \to \tau_2{}^n} \qquad \frac{\Sigma, \Delta; \Gamma \vdash e_1\colon \tau_1 \to \tau_2{}^n \quad \Sigma, \Delta; \Gamma \vdash e_2\colon \tau_1{}^n}{\Sigma, \Delta; \Gamma \vdash e_1\ e_2\colon \tau_2{}^n}$$

$$(\text{fix})\ \frac{\Sigma, \Delta; \Gamma, x\colon \tau^n \vdash e\colon \tau^n}{\Sigma, \Delta; \Gamma \vdash \mathsf{fix}\ x.e\colon \tau^n} \qquad \frac{}{\Sigma, \Delta; \Gamma \vdash \mathsf{z}\colon \mathsf{nat}^n} \qquad \frac{\Sigma, \Delta; \Gamma \vdash e\colon \mathsf{nat}^n}{\Sigma, \Delta; \Gamma \vdash \mathsf{s}\ e\colon \mathsf{nat}^n}$$

$$(\text{case*})\ \frac{\Sigma, \Delta; \Gamma \vdash e\colon \mathsf{nat}^n \quad \Sigma, \Delta; \Gamma \vdash e_1\colon \tau^n \quad \Sigma, \Delta, x\colon \mathsf{nat}^n; \Gamma \vdash e_2\colon \tau^n}{\Sigma, \Delta; \Gamma \vdash (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{z} \to e_1 \mid \mathsf{s}\ x \to e_2)\colon \tau^n}$$

$$\frac{\Sigma, \Delta; \Gamma \vdash e\colon \tau^{n+1}}{\Sigma, \Delta; \Gamma \vdash \langle e\rangle\colon \langle\tau\rangle^n} \qquad \frac{\Sigma, \Delta; \Gamma \vdash e\colon \langle\tau\rangle^n}{\Sigma, \Delta; \Gamma \vdash \tilde{}e\colon \tau^{n+1}} \qquad \frac{\Sigma, \Delta; \Gamma \vdash e\colon [\langle\tau\rangle]^n}{\Sigma, \Delta; \Gamma \vdash \mathsf{run}\ e\colon \tau^n}$$

$$(\text{close})\ \frac{\Sigma, \Delta^{\leq n}; \emptyset \vdash e\colon \tau^n}{\Sigma, \Delta; \Gamma \vdash [e]\colon [\tau]^n} \qquad \frac{\Sigma, \Delta; \Gamma \vdash e_1\colon [\tau_1]^n \quad \Sigma, \Delta, x\colon \tau_1^n; \Gamma \vdash e_2\colon \tau_2{}^n}{\Sigma, \Delta; \Gamma \vdash (\mathsf{let}\ [x] = e_1\ \mathsf{in}\ e_2)\colon \tau_2{}^n}$$

$$\frac{\Sigma, \Delta; \Gamma \vdash e\colon \sigma^n}{\Sigma, \Delta; \Gamma \vdash \mathsf{ref}\ e\colon \mathsf{ref}\ \sigma^n} \qquad \frac{\Sigma, \Delta; \Gamma \vdash e\colon \mathsf{ref}\ \sigma^n}{\Sigma, \Delta; \Gamma \vdash !\ e\colon \sigma^n}$$

$$(\text{set})\ \frac{\Sigma, \Delta; \Gamma \vdash e_1\colon \mathsf{ref}\ \sigma^n \quad \Sigma, \Delta; \Gamma \vdash e_2\colon \sigma^n}{\Sigma, \Delta; \Gamma \vdash e_1 := e_2\colon \mathsf{ref}\ \sigma^n} \qquad \frac{}{\Sigma, \Delta; \Gamma \vdash l\colon \mathsf{ref}\ \sigma^n}\ \Sigma(l) = \mathsf{ref}\ \sigma$$

$$(\text{fix*})\ \frac{\Sigma, \Delta^{\leq n}, x\colon \tau^n; \emptyset \vdash e\colon \tau^n}{\Sigma, \Delta; \Gamma \vdash \mathsf{fix}\ x.e\colon \tau^n} \qquad (\text{close*})\ \frac{\Sigma, \Delta; \Gamma \vdash e\colon \sigma^n}{\Sigma, \Delta; \Gamma \vdash [e]\colon [\sigma]^n}$$

Fig. 2. Type System for $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$

2.1 Type System

Figure 2 gives the rules for the type system of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$. A typing judgement has the form $\Sigma, \Delta; \Gamma \vdash e\colon \tau^n$, read *"e has type τ and level n in $\Sigma, \Delta; \Gamma$"*. Σ gives the type of locations which can be used in e, Δ and Γ (must have disjoint domains and) give the type and level of variables which may occur free in e.

Remark 3. Splitting the context into two parts (Δ and Γ) is borrowed from $\lambda^{\Box}$ [DP96], and allows us to replace the cumbersome closedness annotation ($\mathsf{close}\ e\ \mathsf{with}\ \{x_i = e_i | i \in m\}$) of λ^{BN} [BMTS99] with the more convenient $[e]$ and ($\mathsf{let}\ [x] = e_1\ \mathsf{in}\ e_2$). Informally, a variable $x\colon \tau^n$ declared in Γ ranges over values of type τ at level n (see Definition 1), while a variable $x\colon \tau^n$ declared in Δ ranges over *closed* values (i.e. without free variables) of type τ at level n.

Most typing rules are similar to those for related languages [Dav96,BMTS99], but there are some notable exceptions:

- (close) is the *standard* rule for $[e]$, the restricted context $\Sigma, \Delta^{\leq n}; \emptyset$ in the premise prevents $[e]$ to depend on variables declared in Γ (like in $\lambda^{\Box}$ [DP96]) or variables of level $> n$. The stronger rule (close*) applies only to closed types, and it is *justified* in Remark 5.
- (fix) is the *standard* rule for $\mathsf{fix}\ x.e$, while (fix*) makes a stronger assumption on x, and thus can type recursive definitions (e.g. of closed functions) that

are not typable with (fix). For instance, from $\emptyset; f'\colon [\tau_1 \to \tau_2]^n, x\colon \tau_1^n \vdash e\colon \tau_2{}^n$ we cannot derive $\mathsf{fix}\ f'.[\lambda x.e]\colon [\tau_1 \to \tau_2]^n$, while the following modified term $\mathsf{fix}\ f'.(\mathsf{let}\ [f] = f'\ \mathsf{in}\ [\lambda x.e[f' := [f]]])$ has the right type, but the wrong behaviour (it diverges!). On the other hand, the stronger rule (fix*) allows to type $[\mathsf{fix}\ f.\lambda x.e[f' := [f]]]$, which has the desired operational behaviour.

- There is a *weaker* variant of (case*), which we ignore, where the assumption $x\colon \mathsf{nat}^n$ is in Γ instead of Δ.
- (set) does not assign to $e_1 := e_2$ type unit, simply to avoid adding a unit type to $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$.

The type system enjoys the following basic properties:

Lemma 1 (Weakening).

1. *If* $\Sigma, \Delta; \Gamma \vdash e\colon \tau_2{}^n$ *and* x fresh, *then* $\Sigma, \Delta; \Gamma, x\colon \tau_1^m \vdash e\colon \tau_2{}^n$
2. *If* $\Sigma, \Delta; \Gamma \vdash e\colon \tau_2{}^n$ *and* x fresh, *then* $\Sigma, \Delta, x\colon \tau_1^m; \Gamma \vdash e\colon \tau_2{}^n$
3. *If* $\Sigma, \Delta; \Gamma \vdash e\colon \tau_2{}^n$ *and* l fresh, *then* $\Sigma, l\colon \mathsf{ref}\ \sigma_1, \Delta; \Gamma \vdash e\colon \tau_2{}^n$

Proof. Part 1 is proved by induction on the derivation of $\Sigma, \Delta; \Gamma \vdash e\colon \tau_2{}^n$. The other two parts are proved similarly.

Lemma 2 (Substitution).

1. *If* $\Sigma, \Delta; \Gamma \vdash e\colon \tau_1{}^m$ *and* $\Sigma, \Delta; \Gamma, x\colon \tau_1^m \vdash e'\colon \tau_2{}^n$, *then* $\Sigma, \Delta; \Gamma \vdash e'[x := e]\colon \tau_2{}^n$
2. *If* $\Sigma, \Delta^{\leq m}; \emptyset \vdash e\colon \tau_1{}^m$ *and* $\Sigma, \Delta, x\colon \tau_1^m; \Gamma \vdash e'\colon \tau_2{}^n$, *then* $\Sigma, \Delta; \Gamma \vdash e'[x := e]\colon \tau_2{}^n$

Proof. Part 1 is proved by induction on the derivation of $\Delta; \Gamma, x\colon \tau_1^m \vdash e'\colon \tau_2{}^n$. Part 2 is proved similarly.

2.2 CBV Operational Semantics

Figure 3 gives the evaluation rules for the call-by-value (CBV) operational semantics of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$. Evaluation of a term e at level n can lead to

- a *result* v and a new store μ', when we can derive $\mu, e \overset{n}{\hookrightarrow} \mu', v$,
- a *run-time error*, when we can derive $\mu, e \overset{n}{\hookrightarrow} \mathsf{err}$, or
- *divergence*, when the search for a derivation goes into an *infinite regress*.

We will show that the second case (error) does not occur for well-typed programs (see Theorem 1). In general v ranges over terms, but under appropriate assumptions on μ, v could be restricted to *value at level* n.

Definition 1. *We define the set* $\mathsf{V}^n \subset \mathsf{E}$ *of* **values at level** n *by the BNF*

$$
\begin{aligned}
v^0 \in \mathsf{V}^0 ::= {} & \lambda x.e \mid \mathsf{z} \mid \mathsf{s}\ v^0 \mid \langle v^1 \rangle \mid [v^0] \mid l \\
v^{n+1} \in \mathsf{V}^{n+1} ::= {} & x \mid \lambda x.v^{n+1} \mid v_1^{n+1} v_2^{n+1} \mid \mathsf{fix}\ x.v^{n+1} \mid \\
& \mathsf{z} \mid \mathsf{s}\ v^{n+1} \mid (\mathsf{case}\ v^{n+1}\ \mathsf{of}\ \mathsf{z} \to v_1^{n+1} \mid \mathsf{s}\ x \to v_2^{n+1}) \mid \\
& \langle v^{n+2} \rangle \mid \mathsf{run}\ v^{n+1} \mid [v^{n+1}] \mid (\mathsf{let}\ [x] = v^{n+1}\ \mathsf{in}\ v^{n+1}) \mid \\
& \mathsf{ref}\ v^{n+1} \mid !\ v^{n+1} \mid v_1^{n+1} := v_2^{n+1} \mid l \mid \mathsf{fault} \\
v^{n+2} \in \mathsf{V}^{n+2} + = {} & \tilde{}\, v^{n+1}
\end{aligned}
$$

Normal Evaluation

We give an *exhaustive* set of rules for evaluation of terms $e \in \mathsf{E}$ at level 0

$$\mu, x \overset{0}{\hookrightarrow} \mathsf{err} \qquad \frac{\mu, e_1 \overset{0}{\hookrightarrow} \mu', \lambda x.e \quad \mu', e_2 \overset{0}{\hookrightarrow} \mu'', v \quad \mu'', e[x{:=}\,v] \overset{0}{\hookrightarrow} \mu''', v'}{\mu, e_1\ e_2 \overset{0}{\hookrightarrow} \mu''', v'}$$

$$\mu, \lambda x.e \overset{0}{\hookrightarrow} \mu, \lambda x.e$$

$$\frac{\mu, e_1 \overset{0}{\hookrightarrow} \mu', v \not\equiv \lambda x.e}{\mu, e_1\ e_2 \overset{0}{\hookrightarrow} \mathsf{err}} \qquad \frac{\mu, e[x{:=}\,\mathsf{fix}\ x.e] \overset{0}{\hookrightarrow} \mu', v}{\mu, \mathsf{fix}\ x.e \overset{0}{\hookrightarrow} \mu', v} \qquad \mu, \mathsf{z} \overset{0}{\hookrightarrow} \mu, \mathsf{z} \qquad \frac{\mu, e \overset{0}{\hookrightarrow} \mu', v}{\mu, \mathsf{s}\ e \overset{0}{\hookrightarrow} \mu', \mathsf{s}\ v}$$

$$\frac{\mu, e \overset{0}{\hookrightarrow} \mu', \mathsf{z} \quad \mu', e_1 \overset{0}{\hookrightarrow} \mu'', v}{\mu, (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{z} \to e_1 \mid \mathsf{s}\ x \to e_2) \overset{0}{\hookrightarrow} \mu'', v} \qquad \frac{\mu, e \overset{0}{\hookrightarrow} \mu', v \not\equiv \mathsf{z} \mid \mathsf{s}\ e'}{\mu, (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{z} \to e_1 \mid \mathsf{s}\ x \to e_2) \overset{0}{\hookrightarrow} \mathsf{err}}$$

$$\frac{\mu, e \overset{0}{\hookrightarrow} \mu', \mathsf{s}\ v \quad \mu', e_2[x{:=}\,v] \overset{0}{\hookrightarrow} \mu'', v'}{\mu, (\mathsf{case}\ e\ \mathsf{of}\ \mathsf{z} \to e_1 \mid \mathsf{s}\ x \to e_2) \overset{0}{\hookrightarrow} \mu'', v'} \qquad \frac{\mu, e \overset{1}{\hookrightarrow} \mu', v}{\mu, \langle e \rangle \overset{0}{\hookrightarrow} \mu', \langle v \rangle} \qquad \mu, \tilde{\ } e \overset{0}{\hookrightarrow} \mathsf{err}$$

$$\frac{\mu, e \overset{0}{\hookrightarrow} \mu', [\langle v \rangle] \quad \mu', v \overset{0}{\hookrightarrow} \mu'', v'}{\mu, \mathsf{run}\ e \overset{0}{\hookrightarrow} \mu'', v'} \qquad \frac{\mu, e \overset{0}{\hookrightarrow} \mu', v \not\equiv [\langle e' \rangle]}{\mu, \mathsf{run}\ e \overset{0}{\hookrightarrow} \mathsf{err}} \qquad \frac{\mu, e \overset{0}{\hookrightarrow} \mu', v}{\mu, [e] \overset{0}{\hookrightarrow} \mu', [v]}$$

$$\frac{\mu, e_1 \overset{0}{\hookrightarrow} \mu', [v] \quad \mu', e_2[x{:=}\,v] \overset{0}{\hookrightarrow} \mu'', v'}{\mu, (\mathsf{let}\ [x] = e_1\ \mathsf{in}\ e_2) \overset{0}{\hookrightarrow} \mu'', v'} \qquad \frac{\mu, e_1 \overset{0}{\hookrightarrow} \mu', v \not\equiv [e]}{\mu, (\mathsf{let}\ [x] = e_1\ \mathsf{in}\ e_2) \overset{0}{\hookrightarrow} \mathsf{err}}$$

$$\frac{\mu, e \overset{0}{\hookrightarrow} \mu', v}{\mu, \mathsf{ref}\ e \overset{0}{\hookrightarrow} \mu'\{l = v\}, l}\ l \notin dom(\mu') \qquad \frac{\mu, e \overset{0}{\hookrightarrow} \mu', l}{\mu, !\ e \overset{0}{\hookrightarrow} \mu', v}\ \mu'(l) \equiv v$$

$$\frac{\mu, e \overset{0}{\hookrightarrow} \mu', v \not\equiv l \in dom(\mu')}{\mu, !\ e \overset{0}{\hookrightarrow} \mathsf{err}} \qquad \frac{\mu, e_1 \overset{0}{\hookrightarrow} \mu', l \quad \mu', e_2 \overset{0}{\hookrightarrow} \mu'', v}{\mu, e_1 := e_2 \overset{0}{\hookrightarrow} \mu''\{l = v\}, l}$$

$$\frac{\mu, e_1 \overset{0}{\hookrightarrow} \mu', v \not\equiv l \in dom(\mu')}{\mu, e_1 := e_2 \overset{0}{\hookrightarrow} \mathsf{err}} \qquad \mu, l \overset{0}{\hookrightarrow} \mu, l \qquad \mu, \mathsf{fault} \overset{0}{\hookrightarrow} \mathsf{err}$$

Symbolic Evaluation

$$\frac{\mu, e \overset{0}{\hookrightarrow} \mu', \langle v \rangle}{\mu, \tilde{\ } e \overset{1}{\hookrightarrow} \mu', v} \qquad \frac{\mu, e \overset{0}{\hookrightarrow} \mu', v \not\equiv \langle e' \rangle}{\mu, \tilde{\ } e \overset{1}{\hookrightarrow} \mathsf{err}} \qquad \frac{\mu, e \overset{n+2}{\hookrightarrow} \mu', v}{\mu, \langle e \rangle \overset{n+1}{\hookrightarrow} \mu', \langle v \rangle} \qquad \frac{\mu, e \overset{n+1}{\hookrightarrow} \mu', v}{\mu, \tilde{\ } e \overset{n+2}{\hookrightarrow} \mu', \tilde{\ } v}$$

In all other cases symbolic evaluation is applied to the immediate sub-terms from left to right without changing level $\dfrac{\mu, e_1 \overset{n+1}{\hookrightarrow} \mu', v_1 \quad \mu', e_2 \overset{n+1}{\hookrightarrow} \mu'', v_2}{\mu, e_1\ e_2 \overset{n+1}{\hookrightarrow} \mu'', v_1\ v_2}$ and bound variables that have *leaked* in the store are replaced by fault $\dfrac{\mu, e \overset{n+1}{\hookrightarrow} \mu', v}{\mu, \lambda x.e \overset{n+1}{\hookrightarrow} \mu'[x{:=}\,\mathsf{fault}], \lambda x.v}$

Error Propagation

For space reasons, we omit the rules for error propagation. These rules follow the ML-convention for exceptions propagation.

Fig. 3. Operational Semantics for $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$

Remark 4. Values at level 0 can be classified according to the five kinds of types:

types	$\tau_1 \to \tau_2$	nat	$\langle\tau\rangle$	$[\tau]$	$\mathsf{ref}\ \sigma$
values	$\lambda x.e$	$\mathsf{z},\quad \mathsf{s}\ v^0$	$\langle v^1\rangle$	$[v^0]$	l

Because of $\langle v^1\rangle$ the definition of value at level 0 involves values at higher levels. Values at level > 0, called *symbolic values*, are almost like terms. The differences between the BNF for V^{n+1} and E is in the productions for $\langle e\rangle$ and $\tilde{}e$:

- $\langle v^{n+1}\rangle$ is a value at level n, rather than level $n+1$
- $\tilde{}v^{n+1}$ is a value at level $n+2$, rather than level $n+1$.

Note 2. We will use the following auxiliary notation to describe stores:

- μ is **value store** $\overset{\Delta}{\Longleftrightarrow} \mu\colon \mathsf{L} \overset{fin}{\to} \mathsf{V}^0$;
- $\Sigma \models \mu \overset{\Delta}{\Longleftrightarrow} \mu$ is a value store and $dom(\Sigma) = dom(\mu)$ and $\Sigma; \emptyset \vdash \mu(l)\colon \sigma^0$ whenever $l \in dom(\mu)$.

The following result establishes basic facts about the operational semantics, which are independent of the type system.

Lemma 3 (Values). *$\mu, e \overset{n}{\hookrightarrow} \mu', v$ implies $dom(\mu) \subseteq dom(\mu')$ and $\mathrm{FV}(\mu', v) \subseteq \mathrm{FV}(\mu, e)$; moreover, if μ is a value store, then $v \in \mathsf{V}^n$ and μ' is a value store.*

Proof. By induction on the derivation of the evaluation judgement $\mu, e \overset{n}{\hookrightarrow} \mu', v$.

The following property justifies why a $\sigma \in \mathsf{C}$ is called a closed type.

Lemma 4 (Closedness). *$\Sigma, \Delta^{+1}; \Gamma^{+1} \vdash v^0\colon \sigma^0$ implies $\mathrm{FV}(v^0) = \emptyset$.*

Proof. By induction on the derivation of $\Sigma, \Delta^{+1}; \Gamma^{+1} \vdash v^0\colon \sigma^0$.

Remark 5. Let $\mathsf{V}_\tau \overset{\Delta}{=} \{v \in \mathsf{V}^0 | \Sigma, \Delta^{+1}; \Gamma^{+1} \vdash v\colon \tau^0\}$ be the set of values of type τ (in a given context $\Sigma, \Delta^{+1}; \Gamma^{+1}$). It is easy to show that the mapping $[v] \mapsto v$ is an injection of $\mathsf{V}_{[\tau]}$ into V_τ, and moreover it is *represented* by the term $open \overset{\Delta}{=} \lambda x.(\mathsf{let}\ [x] = x\ \mathsf{in}\ x)$, i.e. $open\colon [\tau] \to \tau$ and $open\ [v] \overset{0}{\hookrightarrow} v$.
Note also that the Closedness Lemma implies the mapping $[v] \mapsto v$ is a bijection when τ is a closed type. A posteriori, this property justifies the typing rule (close*), which in turn ensures that term $close \overset{\Delta}{=} \lambda x.[x]$, representing the inverse mapping $v \mapsto [v]$, has type $\sigma \to [\sigma]$.

Evaluation of Run at level 0 requires to view a value at level 1 as a term to be evaluated at level 0. The following result says that this confusion in the levels is compatible with the type system.

Lemma 5 (Demotion). *$\Sigma, \Delta^{+1}; \Gamma^{+1} \vdash v^{n+1}\colon \tau^{n+1}$ implies $\Sigma, \Delta; \Gamma \vdash v^{n+1}\colon \tau^n$.*

```
datatype  nat
val p = fn : nat -> real -> real ref -> unit

val p_a = fn : nat -> <real> -> <real ref> -> <unit>
val p_cg = fn : nat  -> <real -> real ref -> unit>
val p_sc = <fn x y => (y:=1.0; y:=x*!y; y:=x*!y; y:=x*!y)>
         : <real -> real ref -> unit>

> val p_sp = run[p_sc];                        (* specialised program *)
val p_sp = fn : real -> real ref -> unit

> val p_pg = fn n => let [n]=[n] in run[p_cg n]; (* program generator *)
val p_pg = fn : nat -> real -> real ref -> unit
```

Fig. 4. The Example Written in $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$

Proof. By induction on the derivation of $\Sigma, \Delta^{+1}; \Gamma^{+1} \vdash v^{n+1} : \tau^{n+1}$.

To fully claim the *reflective* nature of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$ we need also a Promotion Lemma (which, however, is not relevant to the proof of Type Safety).

Lemma 6. *$\Sigma, \Delta; \Gamma \vdash e : \tau^n$ implies $e \in \mathsf{V}^{n+1}$ and $\Sigma, \Delta^{+1}; \Gamma^{+1} \vdash e : \tau^{n+1}$.*

Finally, we establish the key result relating the type system to the operational semantics. This result entails that evaluation of a well-typed program $\emptyset; \emptyset \vdash e : \tau^0$ cannot raise an error, i.e. $\emptyset, e \overset{0}{\hookrightarrow} \mathsf{err}$ is not derivable.

Theorem 1 (Safety). *$\mu, e \overset{n}{\hookrightarrow} d$ and $\Sigma \models \mu$ and $\Sigma, \Delta^{+1}; \Gamma^{+1} \vdash e : \tau^n$ imply that there exist μ' and v^n and Σ' such that $d \equiv (\mu', v^n)$ and $\Sigma, \Sigma' \models \mu'$ and $\Sigma, \Sigma', \Delta^{+1}; \Gamma^{+1} \vdash v^n : \tau^n$.*

Proof. By induction on the derivation of the evaluation judgement $\mu, e \overset{n}{\hookrightarrow} d$.

2.3 The Power Function

While ensuring the safety of $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$ requires a relatively non-trivial type system, the power examples presented at the beginning of this paper can still be expressed just as concisely as in MetaML. First, we introduce the following top-level derived forms:

- `val x = e; p` stands for $(\mathsf{let}\ [x] = [e]\ \mathsf{in}\ p)$, with the following derived rules for typing and evaluation at level 0

$$\frac{\Sigma, \Delta^{\leq n}; \emptyset \vdash e : {\tau_1}^n \qquad \Sigma, \Delta, x : \tau_1^n; \Gamma \vdash p : {\tau_2}^n}{\Sigma, \Delta; \Gamma \vdash (\texttt{val}\ x\ \texttt{=}\ e\texttt{;}\ p) : {\tau_2}^n} \qquad \frac{\mu, e \overset{0}{\hookrightarrow} \mu', v \qquad \mu', p[x := v] \overset{0}{\hookrightarrow} \mu'', v'}{\mu, (\texttt{val}\ x\ \texttt{=}\ e\texttt{;}\ p) \overset{0}{\hookrightarrow} \mu'', v'}$$

- a top-level definition by pattern-matching is reduced to one of the form `val f = e; p` in the usual way (that is, using the case and fix constructs).

Note that this means identifiers declared at top-level go in the closed part Δ of a context $\Sigma, \Delta; \Gamma$. We assume to have a predefined closed type real with a function times $*\colon \mathsf{real} \to \mathsf{real} \to \mathsf{real}$ and a constant $1.0\colon \mathsf{real}$. Figure 4 reconsider the example of Figure 1 used in the introduction in $\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$:

- the declarations of `p`, `p_a`, `p_cg` and `p_sc` do not require any change;
- in the declaration of `p_sp` one closedness annotation has been added;
- `p_pg` is a program generator with the same type of the conventional program `p`, but applied to a natural, say 3, returns a specialised program (i.e. `p_sp`).

3 Related Work

The problem we identify at the beginning of this paper also applies to Davies's $\lambda^{\bigcirc}$ [Dav96], which allows open code and symbolic evaluation under lambda (but has no construct for running code). Therefore, the naive addition of references leads to the same problem of scope extrusion pointed out in the Introduction.

$\mathsf{Mini\text{-}ML}^{\mathsf{BN}}_{\mathsf{ref}}$ is related to Binding-Time Analyses (BTAs) for imperative languages. Intuitively, a BTA takes a single-stage program and produces a two-stage one (often in the form of a two-level program) [JGS93,Tah00]. Thiemann and Dussart [TD] describe an off-line partial evaluator for a higher-order language with first-class references, where a two-level language with regions is used to specify a BTA. Their two-level language allows storing dynamic values in static cells, but the type and effect system prohibits operating on static cells within the scope of a dynamic lambda (unless these cells belong to a region local to the body of the dynamic lambda). While both this BTA and our type system ensure that no run-time error (such as scope extrusion) can occur, they provide incomparable extensions.

Hatcliff and Danvy [HD97] propose a partial evaluator for a computational metalanguage, and they formalise existing techniques in a uniform framework by abstracting from dynamic computational effects. However, this partial evaluator does not seem to allow interesting computational effects at specialisation time.

References

BMTS99. Zine El-Abidine Benaissa, Eugenio Moggi, Walid Taha, and Tim Sheard. Logical modalities and multi-stage programming. In *Federated Logic Conference (FLoC) Satellite Workshop on Intuitionistic Modal Logics and Applications (IMLA)*, July 1999.

CDDK86. Dominique Clement, Joelle Despeyroux, Thierry Despeyroux, and Gilles Kahn. A simple applicative language: Mini-ML. In *Proceedings of the 1986 ACM Conference on Lisp and Functional Programming*, pages 13–27. ACM, ACM, August 1986.

Dav96. Rowan Davies. A temporal-logic approach to binding-time analysis. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 184–195, New Brunswick, July 1996. IEEE Computer Society Press.

DP96. Rowan Davies and Frank Pfenning. A modal analysis of staged computation. In *23rd Annual ACM Symposium on Principles of Programming Languages (POPL'96)*, pages 258–270, St. Petersburg Beach, January 1996.

GJ91. Carsten K. Gomard and Neil D. Jones. A partial evaluator for untyped lambda calculus. *Journal of Functional Programming*, 1(1):21–69, January 1991.

GJ96. Robert Glück and Jesper Jørgensen. Fast binding-time analysis for multi-level specialization. In Dines Bjørner, Manfred Broy, and Igor V. Pottosin, editors, *Perspectives of System Informatics*, volume 1181 of *Lecture Notes in Computer Science*, pages 261–272. Springer-Verlag, 1996.

HD97. John Hatcliff and Olivier Danvy. A computational formalization for partial evaluation. *Mathematical Structures in Computer Science*, 7(5):507–541, October 1997.

JGS93. Neil D. Jones, Carsten K Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, 1993.

Met00. The MetaML Home Page, 2000. Provides source code and documentation online at http://www.cse.ogi.edu/PacSoft/projects/metaml/index.html.

Mog98. Eugenio Moggi. Functor categories and two-level languages. In *FoSSaCS '98*, volume 1378 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

MTBS99. Eugenio Moggi, Walid Taha, Zine El-Abidine Benaissa, and Tim Sheard. An idealized MetaML: Simpler, and more expressive. In *European Symposium on Programming (ESOP)*, volume 1576 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag, 1999.

MTHM97. Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

Plo75. Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

Tah99. Walid Taha. *Multi-Stage Programming: Its Theory and Applications*. PhD thesis, Oregon Graduate Institute of Science and Technology, July 1999.

Tah00. Walid Taha. A sound reduction semantics for untyped CBN multi-stage computation. Or, the theory of MetaML is non-trivial. In *Proceedings of the ACM Symposium on Partial Evaluation and Semantics Based Program Manipulation*, Boston, January 2000.

TBS98. Walid Taha, Zine-El-Abidine Benaissa, and Tim Sheard. Multi-stage programming: Axiomatization and type-safety. In *25th International Colloquium on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 918–929, Aalborg, July 1998.

TD. Peter Thiemann and Dirk Dussart. Partial evaluation for higher-order languages with state. Available online from http://www.informatik.uni-freiburg.de/~thiemann/papers/index.html.

TS97. Walid Taha and Tim Sheard. Multi-stage programming with explicit annotations. In *Proceedings of the ACM-SIGPLAN Symposium on Partial Evaluation and semantic based program manipulations PEPM'97, Amsterdam*, pages 203–217. ACM, 1997.

TS00. Walid Taha and Tim Sheard. MetaML: Multi-stage programming with explicit annotations. *Theoretical Computer Science*, 248(1-2), 2000.

A Statically Allocated Parallel Functional Language

Alan Mycroft[1,2] and Richard Sharp[2]

[1] Computer Laboratory, Cambridge University
New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK
am@cl.cam.ac.uk
[2] AT&T Laboratories Cambridge
24a Trumpington Street, Cambridge CB2 1QA, UK
rws@uk.research.att.com

Abstract. We describe SAFL, a call-by-value first-order functional language which is syntactically restricted so that storage may be statically allocated to fixed locations. Evaluation of independent sub-expressions happens in parallel—we use locking techniques to protect shared-use function definitions (i.e. to prevent unrestricted parallel accesses to their storage locations for argument and return values). SAFL programs have a well defined notion of total (program and data) size which we refer to as 'area'; similarly we can talk about execution 'time'. Fold/unfold transformations on SAFL provide mappings between different points on the area-time spectrum. The space of functions expressible in SAFL is incomparable with the space of primitive recursive functions, in particular interpreters are expressible. The motivation behind SAFL is hardware description and synthesis—we have built an optimising compiler for translating SAFL to silicon.

1 Introduction

This paper addresses the idea of a functional language, SAFL, which

- can be statically allocated—all variables are allocated to fixed storage locations at compile time—there is no stack or heap; and
- has independent sub-expressions evaluated concurrently.

While this concept might seem rather odd in terms of the capabilities of modern processor instruction sets, our view is that it neatly abstracts the primitives available to a hardware designer. Our desire for static allocation is motivated by the observation that dynamically-allocated storage does not map well onto silicon: an addressable global store leads to a von Neumann bottleneck which inhibits the natural parallelism of a circuit. SAFL has a call-by-value semantics since strict evaluation naturally facilitates parallel execution which is well suited to hardware implementation.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 37–48, 2000.
© Springer-Verlag Berlin Heidelberg 2000

To emphasise the hardware connection we define the *area* of a SAFL program to be the total space required for its execution. Due to static allocation we see that *area* is $O(\textit{length of program})$; similarly we can talk about execution *time*. Fold/unfold transformations [1] at the SAFL level correspond directly to area-time tradeoffs at the hardware level.

In this paper we are concerned with the properties of the SAFL language itself rather than the details of its translation to hardware. In the light of this and for the sake of clarity, we present an implementation of SAFL by means of a translation to an abstract machine code which we claim mirrors the primitives available in hardware. The design of an optimising compiler which translates SAFL into hardware is presented in a companion paper [11]. A more practical use of SAFL for hardware/software co-design is given in [9].

The body of this paper is structured as follows. Section 2 describes the SAFL language and Section 3 describes an implementation on a parallel abstract machine. In Sections 4 and 5 we argue that SAFL is well suited for hardware description and synthesis. Section 6 shows how fold/unfold transformations can represent SAFL area-time tradeoffs. Finally, Sections 7 and 8 discuss more theoretical issues: how SAFL relates to Primitive Recursive functions and problems concerning higher-order extensions. Section 9 concludes and outlines some future directions.

Comparison with Other Work

The motivation for static allocation is not new. Gomard and Sestoft [2] describe *globalization* which detects when stack or heap allocation of function parameters can be implemented more efficiently with global variables. However, whereas globalization is an optimisation which may in some circumstances improve performance, in our work static allocation is a fundamental property of SAFL enforced by the syntactic restrictions described in Section 2.

Previous work on compiling declarative specifications to hardware has centred on how functional languages themselves can be used as tools to aid the design of circuits. Sheeran's muFP [12] and Lava [13] systems use functional programming techniques (such as higher order functions) to express concisely the repeating structures that often appear in hardware circuits. In this framework, using different interpretations of primitive functions corresponds to various operations including behavioural simulation and netlist generation. Our approach takes SAFL constructs (rather than gates) as primitive. Although this restricts the class of circuits we can describe to those which satisfy certain high-level properties, it permits high-level analysis and optimisation yielding efficient hardware. We believe our association of function definitions with hardware resources (see Section 4) to be novel.

Various authors have described silicon compilers (e.g. for C [4] and Occam [10]). Although rather beyond the scope of this paper, we argue that the flexibility of functional languages provides much more scope for analysis and optimisation.

Hofmann [6] describes a type system which allows space pre-allocated for argument data-structures to be re-used by in-place update. Boundedness there means that no new heap space is allocated although stack space may be unbounded. As such our notion of static allocatability is rather stronger.

2 Formalism

We use a first-order language of recursion equations (higher order features are briefly discussed in Section 8). Let c range over a set of constants, x over variables (occurring in `let` declarations or as formal parameters), a over primitive functions (such as addition) and f over user-defined functions. For typographical convenience we abbreviate formal parameter lists $(x_1, \ldots, x_k)$ and actual parameter lists $(e_1, \ldots, e_k)$ to $\vec{x}$ and $\vec{e}$ respectively; the same abbreviations are used in `let` definitions. SAFL has syntax of:

– terms e given by:

$$e ::= c \mid x \mid \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 \mid \texttt{let}\ \vec{x} = \vec{e}\ \texttt{in}\ e_0 \mid \\ a(e_1, \ldots, e_{arity(a)}) \mid f(e_1, \ldots, e_{arity(f)})$$

– programs p given by:

$$p ::= \texttt{fun}\ f^{11}(\vec{x}) = e_{11}\ \texttt{and} \ldots \texttt{and}\ f^{1r_1}(\vec{x}) = e_{1r_1} \\ \ldots \\ \texttt{fun}\ f^{n1}(\vec{x}) = e_{n1}\ \texttt{and} \ldots \texttt{and}\ f^{nr_n}(\vec{x}) = e_{nr_n}.$$

We refer to a phrase of the form

$$\texttt{fun}\ f^{i1}(\vec{x}) = e_{i1}\ \texttt{and} \ldots \texttt{and}\ f^{ir_i}(\vec{x}) = e_{ir_i}$$

as a (mutually recursive) function *group*. The notation f^{ij} just means the jth function of group i. Programs have a distinguished function `main` (normally f^{nr_n}) which represents an external world interface—at the hardware level it accepts values on an input port and may later produce a value on an output port.

To simplify semantic descriptions we will further assume that all function and variable names are distinct; this is particularly useful for static allocation since we can use the name of the variable for the storage location to which it is allocated.

We impose additional stratification restrictions[1] on the e_{ij} occurring as bodies of the f^{ij}; arbitrary calls to previous definitions are allowed, but recursion (possibly mutual) is restricted to tail recursion to enforce static allocatability. This is formalised as a well-formedness check. Define the *tailcall contexts*, $\mathcal{TC}$ by

$$\mathcal{TC} ::= [\,] \mid \texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ \mathcal{TC} \mid \texttt{if}\ e_1\ \texttt{then}\ \mathcal{TC}\ \texttt{else}\ e_3 \\ \mid \texttt{let}\ \vec{x} = \vec{e}\ \texttt{in}\ \mathcal{TC}$$

[1] Compare this with *stratified negation* in the deductive database world.

The well-formedness condition is then that, for every user-function application $f^{ij}(\vec{e})$ within function definition $f^{gk}(\vec{x}) = e_{gk}$ in group g, we have that:

$$i < g \vee (i = g \wedge \exists (C \in \mathcal{TC}).e_{gk} = C[f^{ij}(\vec{e})])$$

The first part ($i < g$) is merely static scoping (definitions are in scope if previously declared) while the second part says that a call to a function in the same group (i) as its definition (g) is only valid if the call is actually in tailcall context.

3 Implementing SAFL

We give a translation $[\![\cdot]\!]$ of SAFL programs into an abstract machine code which mirrors primitives available in hardware (its correctness relies on SAFL restrictions). Each function definition corresponds to one block of code. In order to have temporaries available we will assume that each expression and subexpression is labelled by a unique label number (or 'occurrence') from which a storage location can be generated. Label names are assumed to be distinct from variables so we can use the notation M_x and M_ℓ to mean the storage location associated with variable x or label ℓ respectively. We use the notation $\ell : e$ to indicate expression e has label ℓ. The expression

```
if x=1 then y else f(x,y-1)
```

might then be more fully written as (temporarily using the notation e^ℓ instead of $\ell : e$ used elsewhere):

$$\texttt{(if (x}^{\ell 21}\texttt{=1}^{\ell 22}\texttt{)}^{\ell 2}\ \texttt{then y}^{\ell 3}\ \texttt{else (f(x}^{\ell 41}\texttt{,(y}^{\ell 421}\texttt{-1}^{\ell 422}\texttt{)}^{\ell 42}\texttt{)}^{\ell 4}\texttt{)}^{\ell 1}$$

We write $f\texttt{.formals}$ to stand for $(M_{x_1}, \ldots, M_{x_k})$ where $\vec{x}$ is the tuple of formal parameters of f (which are already assumed to be globally distinct from all other variable names). Similarly, we will assume all functions f^{ij} in group i leave their result in the storage location $M_{f^{i*}\texttt{.result}}$—this this is necessary to ensure tailcalls to other functions in group i behave as intended.[2] (The notation $i*$ in general refers to a common resource shared by the members of function group i.)

In addition to the storage location as above, we need two other forms of storage: for each function group i we have a location L_{i*} to store the return link (accessed by *JSR* and *RET*); and a semaphore S_{i*} to protect its (statically allocated) arguments, temporaries and the like from calls in competing *PAR* threads by enforcing mutual exclusion.

The abstract instructions for the machine are as follows:

[2] A type system would require that the result type of all functions in a group are identical—because they return each others' values—so $M_{f^{i*}\texttt{.result}}$ has a well-defined size.

$m := m'$	Copy m' to m.
$(m_1, \ldots, m_k) := (m'_1, \ldots m'_k)$	Copy the m'_i to the m_i in any order.
$m := PRIMOP_a(m_1, \ldots m_k)$	Perform the operation corresponding to built-in primitive a.
$LOCK(S)$	Lock semaphore S.
$UNLOCK(S)$	Release semaphore S.
$JMP(ep)$	Branch to function entrypoint ep—used for tail-call.
$JSR(m, ep)$	Store some representation of the point of call in location m and branch to function entrypoint ep.
$RET(m)$	Branch to the instruction following the point of call specified by m.
$COND(m, seq_1, seq_2)$	If location m holds a 'true' value then execute opcode sequence seq_1 otherwise seq_2.
$PAR(seq_1, \ldots, seq_k)$	Execute opcode sequences $seq_1, \ldots, seq_k$ in parallel, waiting for all to complete before terminating.

Instructions are executed sequentially, except that *JSR*, *JMP* and *RET* alter the execution sequence. The *PAR* construct represents fork-join parallelism (each of the operand sequences are executed) and *COND* the usual conditional (one of the two operand sequences is executed).

Assuming e is a sub-expression of a function body of group g the compilation function $[\![e]\!]^g m$ gives an opcode sequence which evaluates e to storage location m (we omit g for readability in the following—it is only used to identify tailcalls):

$$
\begin{aligned}
&[\![c]\!]m = m := c \\
&[\![x]\!]m = m := M_x \\
&[\![\texttt{if}\ (\ell : e_1)\ \texttt{then}\ e_2\ \texttt{else}\ e_3]\!]m = \\
&\qquad [\![e_1]\!]M_\ell; \\
&\qquad COND(M_\ell, [\![e_2]\!]m, [\![e_3]\!]m) \\
&[\![\texttt{let}\ (x_1, \ldots, x_k) = (e_1, \ldots, e_k)\ \texttt{in}\ e_0]\!]m = \\
&\qquad PAR([\![e_1]\!]M_{x_1}, \ldots, [\![e_k]\!]M_{x_k}); \\
&\qquad [\![e_0]\!]m \\
&[\![a(\ell_1 : e_1, \ldots, \ell_k : e_k)]\!]m = \\
&\qquad PAR([\![e_1]\!]M_{\ell_1}, \ldots, [\![e_k]\!]M_{\ell_k}); \\
&\qquad m := PRIMOP_a(M_{\ell_1}, \ldots, M_{\ell_k}) \\
&[\![f^{ij}(\ell_1 : e_1, \ldots, \ell_k : e_k)]\!]m = \\
&\qquad \left\{ \begin{array}{l} PAR([\![e_1]\!]M_{\ell_1}, \ldots, [\![e_k]\!]M_{\ell_k}); \\ LOCK(S_{i*}); \\ M_{f^{ij}.\texttt{formals}} := (M_{\ell_1}, \ldots, M_{\ell_k}); \\ JSR(L_{i*}, EntryPt_{ij}); \\ m := M_{f^{i*}.\texttt{result}}; \\ UNLOCK(S_{i*}) \end{array} \right\} \text{ if } i < g \\
&[\![f^{ij}(\ell_1 : e_1, \ldots, \ell_k : e_k)]\!]m = \\
&\qquad \left\{ \begin{array}{l} PAR([\![e_1]\!]M_{\ell_1}, \ldots, [\![e_k]\!]M_{\ell_k}); \\ M_{f^{ij}.\texttt{formals}} := (M_{\ell_1}, \ldots, M_{\ell_k}); \\ JMP(EntryPt_{ij}) \end{array} \right\} \text{ if } i = g
\end{aligned}
$$

Finally we need to compile each function definition to a sequence of instructions labelled with the function name:

$$[\![\texttt{fun}\ f^{ij}(x_1,\ldots,x_k) = e_{ij}]\!] = \left\{ \begin{array}{l} EntryPt_{ij}: \\ \quad [\![e_{ij}]\!]^i M_{f^{i*}.\texttt{result}}; \\ \quad RET(L_{i*}) \end{array} \right\}$$

The above translation is naïve (often semaphores and temporary storage locations are used unnecessarily) but explains various aspects; an optimising compiler for hardware purposes has been written [11].

Proposition 1. *The translation* $[\![e]\!]$ *correctly implements SAFL programs in that executing the abstract machine code coincides with standard eager evaluation of* e.

Note that the translation would fail if we use one semaphore-per-function instead of one-per-group. Consider the program

```
fun f(x) = if x=0 then 1 else g(x-1)
and g(x) = if x=0 then 2 else f(x-1);
fun h(x,y) = x+y;
fun main() = h(f(8),g(9));
```

where there is then the risk that the *PAR* construct for the actual arguments to `h` will simultaneously take locks on the semaphores for `f` and `g` resulting in deadlock.

4 Hardware Synthesis Using SAFL

As part of the FLaSH project (Functional Languages for Synthesising Hardware) [11], we have implemented an optimising silicon compiler which translates SAFL specifications into structural Verilog. We have found that SAFL is able to express a wide range of hardware designs; our tools have been used to build a small commercial processor.[3]

The static allocation properties of SAFL allow our compiler to enforce a direct mapping between a function definition:

$$f(\vec{x}) = e$$

and a hardware block, H_f, with output port, P_f, consisting of:

- a fixed amount of storage (registers holding values of the arguments $\vec{x}$) and
- a circuit to compute e to P_f.

Hence, multiple calls to a function f at the source level corresponds directly to sharing the resource H_f at the hardware level. As the FLaSH compiler synthesises multi-threaded hardware, we have to be careful to ensure that multiple

[3] We implemented the instruction set of the Cambridge Consultants XAP processor: `http://www.camcon.co.uk`; we did not support the SIF instruction.

accesses to a shared hardware resource will not occur simultaneously. We say that resource, H_f, is subject to a *sharing conflict* if multiple accesses may occur concurrently. Sharing conflicts are dealt with by inserting arbiters (cf. semaphores in our software translation). Program analysis is used to detect potential sharing conflicts—arbiters are only synthesised where necessary.

Our practical experience of using the FLaSH system to design and build real hardware has brought to light many interesting techniques that conventional hardware description languages cannot exploit. These are outlined below.

4.1 Automatic Generation of Parallel Hardware

Hammond [5] observes:

> "It is almost embarrassingly easy to partition a program written in a strict [functional] language [into parallel threads]. Unfortunately, the partition that results often yields a large number of very fine-grained tasks."

He uses the word *unfortunately* because his discussion takes place in the context of software, where fairly course-grained parallelism is required to ensure the overhead of `fork/join` does not outweigh the benefits of parallel evaluation.

In contrast, we consider the existence of "a large number of very fine-grained tasks" to be a very *fortunate* occurrence: in a silicon implementation, very fine-grained parallelism is provided with virtually no overhead! The FLaSH compiler produces hardware where all function arguments and `let`-declarations are evaluated in parallel.

4.2 Source-Level Program Transformation

We have found that source-level program transformation of SAFL specifications is a powerful technique. A designer can explore a wide range of hardware implementations by repeatedly transforming an initial specification.

We have investigated a number of transformations which correspond to concepts in hardware design. Due to space constraints we can only list the transformations here:

Resource Sharing vs Duplication: Since a single user-defined function corresponds to a single hardware block SAFL provides fine-grained control over resource sharing/duplication.

Static vs Dynamic Scheduling: By default, function arguments are evaluated in parallel. Thus compiling `f(4)+f(5)` will generate an arbiter to sequentialise access to the shared resource $H_{\mathtt{f}}$. Alternatively we can use a `let`-declaration to specify an ordering statically. The circuit corresponding to `let x=f(4) in x+f(5)` does not require dynamic arbitration; we have specified a static order of access to $H_{\mathtt{f}}$.

Area-Time Tradeoffs: We observe that fold/unfold transformations correspond directly to area-time tradeoffs at the hardware level. This can be seen as a generalisation of resource sharing/duplication (see Section 6).

Hardware-Software Partitioning: We have demonstrated [9] a source-source transformation which allows us to represent hardware/software partitioning within SAFL.

At the SAFL level it is relatively straightforward to apply these transformations. Investigating the same tradeoffs entirely within RTL Verilog would require time-consuming and error-prone modifications throughout the code.

4.3 Static Analysis and Optimisation

Declarative languages are more susceptible to analysis and transformation than imperative languages. In order to generate efficient hardware, the FLaSH compiler performs the following high-level analysis techniques (documented in [11]):

Parallel Conflict Analysis is performed at the abstract syntax level, returning a set of function calls which require arbitration at the hardware level.

Register Placement is the process of inserting temporary storage registers into a circuit. The FLaSH compiler translates specifications into intermediate code (based on control/data flow graphs) and performs data-flow analysis at this level in order to place registers. (This optimisation is analogous to minimising the profligate use of M_ℓ temporaries seen in the software translation $[\![\cdot]\!]$.)

Timing Analysis (with respect to a particular implementation strategy) is performed through an abstract interpretation where functions and operators return the times taken to compute their results.

4.4 Implementation Independence

The high level of specification that SAFL provides means that our hardware descriptions are implementation independent. Although the current FLaSH compiler synthesises hardware in a particular style,[4] there is the potential to develop a variety of back-ends, targeting a wide range of hardware implementations.

In particular, we believe that SAFL would lend itself to asynchronous circuit design as the compositional properties of functions map directly onto the compositional properties of asynchronous hardware modules. We plan to design an asynchronous back-end for FLaSH in order to compare synchronous and asynchronous implementations.

5 A Hardware Example

In order to provide a concrete example of the benefits of designing hardware in SAFL consider the following specification of a shift-add multiplier:

```
fun mult(x, y, acc) =
  if (x=0 | y=0) then acc
     else mult(x<<1, y>>1, if y.bit0 then acc+x else acc)
```

[4] The generated hardware is synchronous with bundled data and ready signals.

From this specification, the FLaSH compiler generates a hardware resource, $H_{\texttt{mult}}$, with three data-inputs: (`x`, `y` and `acc`), a data-output (for returning the function result), a control-input to trigger computation and a control-output which signals completion. The multiplier circuit contains some control logic, two 1-place shifters, an adder and three registers which are used to latch data-inputs. We trigger $H_{\texttt{mult}}$ by placing argument values on the data-inputs and signalling an event on the control-input. The result can be read from the data-output when a completion event is signalled on the control-output.

These 3 lines of SAFL produce over 150 lines of RTL Verilog. Synthesising a 16-bit version of `mult`, using Mentor Graphics' *Leonardo* tool, yields 1146 2-input equivalent gates.[5] Implementing the same algorithm directly in RTL Verilog took longer to write and yielded an almost identical gate count.

6 Fold/Unfold for Area-Time Tradeoff

In section 4.2 we observed that the fold/unfold transformation [1] can be used to trade area for time. As an example of this consider:

```
fun f x = ...
fun main(x,y) = g(f(x),f(y))
```

The two calls to `f` are serialised by mutual exclusion before `g` is called. Now use fold/unfold to duplicate `f` as `f'`, replacing the second call to `f` with one to `f'`. This can be done using an unfold, a definition rule and a fold yielding

```
fun f  x = ...
fun f' x = ...
fun main(x,y) = g(f(x),f'(y))
```

The second program has more area than the original (by the size of `f`) but runs more quickly because the calls to `f(x)` and `f'(y)` execute in parallel.

Although the example given above is trivial, we find fold/unfold to be a useful technique in choosing a hardware implementation of a given specification. Note that fold/unfold allows us to do more than resource/duplication sharing tradeoffs. For example, folding/unfolding recursive function calls before compiling to synchronous hardware corresponds to trading the amount of work done per clock cycle against clock speed—`mult` can be mechanically transformed into:

```
fun mult(x, y, acc) =
  if (x=0 | y=0) then acc
  else let (x',y',acc') = (x<<1, y>>1,
                           if y.bit0 then acc+x else acc) in
    if (x'=0 | y'=0) then acc'
    else mult(x'<<1, y'>>1, if y'.bit0 then acc'+x' else acc')
```

which takes half as many clock cycles.

[5] This figure includes the gates required for the three argument registers.

7 Theoretical Expressibility

Here we consider the expressibility of programs in SAFL. Clearly in one sense, each such program represents a finite state machine as it has a bounded number of states and memory locations, and therefore is very inexpressive (but this argument catches the essence of the problem no more than observing that a personal computer is also a finite state automaton).

Consider the relation to Primitive Recursive (PR) functions. In the PR definition scheme, suppose g and h are already shown to be primitive recursive functions (of k and $k+1$ arguments), then the definition of f

$$f(0, x_1, \ldots, x_k) = g(x_1, \ldots, x_k)$$
$$f(n+1, x_1, \ldots, x_k) = h(f(n, x_1, \ldots, x_k), x_1, \ldots, x_k)$$

is also primitive recursive of $k+1$ arguments.[6] We see that the SAFL restrictions require h to be the identity projection on its first argument, but that our definitional scheme allows recursion more general than descent through a well-founded order ($n+1$ via n eventually to 0 in the integer form above). In particular SAFL functions may be partial.

Thus we conclude that, in practice, our statically allocatable functions represent an incomparable subset of general recursion than that subset specified by primitive recursion. Note that we cannot use translation to continuation-passing form where all calls are tailcalls because SAFL is first order (but see the next section). Jones [7] shows the subtle variance of expressive power on recursion forms, assignability and higher-order types.

As a slightly implausible aside, suppose we consider statically allocatable functional languages where values can range over any natural number. In this case the divergence from primitive recursion becomes even clearer—even if we have an assertion that the statically allocated functional program is total then we cannot in general transform it into primitive recursive form. To see this observe that we can code a register machine interpreter as such a statically allocated program with register machine program being Ackermann's function.

8 Higher Order Extensions to SAFL

Clearly a simple addition of higher-order functions to SAFL would break static allocatability by allowing recursion other than in the program structure. Consider for example the traditional

```
let g(n,h) = if n=0 then 1 else n * h(n-1,h)
let f(n) = g(n,g)
```

trick to encode the factorial function in a form which requires $O(n)$ space.[7]

[6] In practice we widen this definition to allow additional intensional forms without affecting the space of functions definable.

[7] Of course one could use an accumulator argument to implement this in $O(1)$ space, but we want the statically allocatability rules to be intensional.

A simple extension is to allow functions to be passed as values, and function valued expressions to be invoked. (No closures can be constructed because of the recursion equation syntax). One implementation of this is as follows: use a control-flow analysis such as 0-CFA to identify possible targets of each *indirect call* (i.e. call to a function-valued expression). This can be done in cubic time. We then fault any indirect calls which are incompatible with the function definition hierarchy in the original program—i.e. those with a function f^{ij} containing a possible indirect call to f^{gk} where $g \geq i$ (unless the indirect call is in tailcall context and $g = i$). Information from 0-CFA also permits a source-to-source transformation to first-order using a case branch over the possible functions callable at that point: we map the call $e'(\vec{e})$ where e' can evaluate to `g`, `h` or `i` into:

`if` e'`=g then g(`$\vec{e}$`) else if` e'`=h then h(`$\vec{e}$`) else i(`$\vec{e}$`)`

The problem with adding nested function definitions (or λ-expressions) is that it is problematic to statically allocate storage for the resulting closures. Even programs which use only tail-recursion and might at first sight appear harmless, such as

```
fun r(x) = ⟨some function depending on x⟩
fun f(x,g) = if x=0 then g(0)
                    else f(x-1, r(x) ∘ g)
```

require unlimited store. Similarly, the translation to CPS (continuation passing style) transforms any program into an equivalent one using only tailcalls, but at the cost of increasing it to higher-order—again see [7] for more details.

One restriction which allows function closures is the Algol solution: functions can be passed as parameters but not returned as results (or at least not beyond the scope any of their free variables). It is well known that such functions can be stack implemented, which in the SAFL world means their storage is bounded. Of course we still need the 0-CFA check as detailed above.

9 Conclusions and Further Work

This paper introduces the idea of statically allocated functional languages which are interesting in themselves as well as being apposite and powerful for expressing hardware designs. However there remains much to be done to explore their uses as hardware synthesis languages, e.g. optimising hardware compilation, type systems, synchronous versus asynchronous translations, etc.

Currently programs support a 'start and wait for result' interface. We realise that in real hardware systems we need to interact with other devices having internal state. We are considering transactional models for such interfaces including the use of channels. Forms of functional language input/output explored in Gordon's thesis [3] may be also be useful.

Acknowledgments

We are indebted to Neil Jones, Martin Hofmann, Simon Peyton-Jones and the anonymous referees for comments which have improved this paper.

References

1. Burstall, R.M. and Darlington, J. A Transformation System for Developing Recursive Programs, JACM 24(1), 1977.
2. Gomard, C.K. and Sestoft, P. Globalization and Live Variables. Proceedings of the Symposium on Partial Evaluation and Semantics-Based Program Manipulation, pp. 166-177. ACM Press. 1991. SIGPLAN NOTICES, vol. 26, number 9.
3. Gordon, A.D., Functional Programming and Input/Output. Distinguished Dissertations in Computer Science. Cambridge University Press, 1994.
4. Greaves, D.J. An Approach Based on Decidability to Compiling C and Similar, Block-Structured, Imperative Languages for Hardware Software Codesign. Unpublished memo, 1999.
5. Hammond, K. Parallel Functional Programming: An Introduction, International Symposium on Parallel Symbolic Computation, World Scientific, 1994.
6. Hofmann, M. A Type System for Bounded Space and Functional In-Place Update. Proc. ESOP'2000 Berlin, Springer-Verlag LNCS, 2000.
7. Jones, N.D. The Expressive Power of Higer-order Types or, Life without CONS. J. Functional Programming, to appear.
8. Milner, R., Tofte, M., Harper, R. and MacQueen, D. The Definition of Standard ML (Revised). MIT Press, 1997.
9. Mycroft, A. and Sharp, R.W. Hardware/Software Co-Design using Functional Languages. Submitted for publication.
10. Page, I. and Luk, W. Compiling Occam into Field-Programmable Gate Arrays. In Moore and Luk (eds.) FPGAs, pages 271-283. Abingdon EE&CS Books, 1991.
11. Sharp, R.W. and Mycroft, A. The FLaSH Compiler: Efficient Circuits from Functional Specifications. Technical Report tr.2000.3, AT&T Laboratories Cambridge. Available from: `www.uk.research.att.com`
12. Sheeran, M. muFP, a Language for VLSI Design. Proc. ACM Symp. on LISP and Functional Programming, 1984.
13. Bjesse, P., Claessen, K., Sheeran, M. and Singh, S. Lava: Hardware Description in Haskell. Proceedings of the 3rd ACM SIGPLAN International Conference on Functional Programming, 1998.

An Optimal Minimum Spanning Tree Algorithm*

Seth Pettie and Vijaya Ramachandran

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
seth@cs.utexas.edu, vlr@cs.utexas.edu

Abstract. We establish that the algorithmic complexity of the minimum spanning tree problem is equal to its decision-tree complexity. Specifically, we present a deterministic algorithm to find a minimum spanning forest of a graph with n vertices and m edges that runs in time $O(\mathcal{T}^*(m,n))$ where $\mathcal{T}^*$ is the minimum number of edge-weight comparisons needed to determine the solution. The algorithm is quite simple and can be implemented on a pointer machine.

Although our time bound is optimal, the exact function describing it is not known at present. The current best bounds known for $\mathcal{T}^*$ are $\mathcal{T}^*(m,n) = \Omega(m)$ and $\mathcal{T}^*(m,n) = O(m \cdot \alpha(m,n))$, where α is a certain natural inverse of Ackermann's function.

Even under the assumption that $\mathcal{T}^*$ is super-linear, we show that if the input graph is selected from $G_{n,m}$, our algorithm runs in linear time w.h.p., regardless of n, m, or the permutation of edge weights. The analysis uses a new martingale for $G_{n,m}$ similar to the edge-exposure martingale for $G_{n,p}$.

Keywords: Graph algorithms; minimum spanning tree; optimal complexity.

1 Introduction

The minimum spanning tree (MST) problem has been studied for much of this century and yet despite its apparent simplicity, the problem is still not fully understood. Graham and Hell [GH85] give an excellent survey of results from the earliest known algorithm of Borůvka [Bor26] to the invention of Fibonacci heaps, which were central to the algorithms in [FT87,GGST86]. Chazelle [Chaz97] presented an MST algorithm based on the Soft Heap [Chaz98] having complexity $O(m\alpha(m,n)\log\alpha(m,n))$, where α is a certain inverse of Ackermann's function. Recently Chazelle [Chaz99] modified the algorithm in [Chaz97] to bring down the running time to $O(m \cdot \alpha(m,n))$. Later, and in independent work, a similar

* Part of this work was supported by Texas Advanced Research Program Grant 003658-0029-1999. Seth Pettie was also supported by an MCD Fellowship.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 49–60, 2000.
© Springer-Verlag Berlin Heidelberg 2000

algorithm of the same running time was presented in Pettie [Pet99], which gives an alternate exposition of the $O(m \cdot \alpha(m,n))$ result. This is the tightest time bound for the MST problem to date, though not known to be optimal.

All algorithms mentioned above work on a pointer machine [Tar79] under the restriction that edge weights may only be subjected to binary comparisons. If a more powerful model is assumed, the MST can be computed optimally. Fredman and Willard [FW90] showed that on a unit-cost RAM where the bit-representation of edge weights may be manipulated, the MST can be computed in linear time. Karger et al. [KKT95] presented a *randomized* MST algorithm that runs in linear time with high probability, even if edge weights are only subject to comparisons.

It is still unknown whether these more powerful models are necessary to compute the MST in linear time. However, we give a deterministic, comparison-based MST algorithm that runs on a pointer machine in $O(\mathcal{T}^*(m,n))$ time, where $\mathcal{T}^*(m,n)$ is the number of edge-weight comparisons needed to determine the MST on any graph with m edges and n vertices. Additionally, by considering our algorithm's performance on random graphs, we show that it runs in linear time w.h.p., regardless of edge-density or the permutation of edge weights.

Although our algorithm is optimal, its precise running time is not known at this time. In view of recent results we can state that the running time of our algorithm is $O(m \cdot \alpha(m,n))$. Clearly, its running time is also $\Omega(m)$.

In the next section we review some well-known MST results that are used by our algorithm. In section 3 we prove a key lemma and give a procedure for partitioning the graph in an MST-respecting manner. Section 4 gives an overview of the optimal algorithm. Section 5 gives the algorithm and a proof of optimality. In section 6 we show our algorithm runs in linear-time w.h.p. if the input graph is selected at random. Sections 7 & 8 discuss related problems, open questions, and the actual complexity of MST.

2 Preliminaries

The input is an undirected graph $G = (V, E)$ where each edge is assigned a distinct real-valued *weight*. The *minimum spanning forest (MSF)* problem asks for a spanning acyclic subgraph of G having the least total weight. Throughout the paper m and n denote the number of edges and vertices in the graph.

It is well-known that one can identify edges provably in the MSF using the *cut* property, and edges provably not in the MSF using the *cycle* property. The cut property states that the lightest edge crossing any partition of the vertex set into two parts must belong to the MSF. The cycle property states that the heaviest edge in any cycle in the graph cannot be in the MSF.

2.1 Boruvka Steps

The earliest known MSF algorithm is due to Borůvka [Bor26]. It proceeds in a sequence of stages, and in each stage it executes a *Borůvka step* on the graph G, which identifies the set F consisting of the minimum-weight edge incident on each vertex in G, includes these edges to the MSF (by the cut property), and then forms the graph $G_1 = G \backslash F$ as the input to the next stage, where

$G \backslash F$ is the graph obtained by contracting each connected component formed by F. This computation can be performed in linear time. Since the number of vertices reduces by at least a factor of two, the running time of this algorithm is $O(m \log n)$.

Our optimal algorithm uses a procedure called Boruvka2$(G; F, G')$. This procedure executes two Boruvka steps on the input graph G and returns the contracted graph G' as well as the set of edges F identified for the MSF during these two steps.

2.2 Dijsktra-Jarník-Prim Algorithm

Another early MSF algorithm that runs in $O(m \log n)$ time is the one by Jarník [Jar30], re-discovered by Dijkstra [Dij59] and Prim [Prim57]. We will refer to this algorithm as the *DJP* algorithm. Briefly, the DJP algorithm grows a tree T, which initially consists of an arbitrary vertex, one edge at a time, choosing the next edge by the following simple criterion: Augment T with the minimum weight edge (x, y) such that $x \in T$ and $y \notin T$. By the cut property, all edges in T are in the MSF. We omit the proof of the following simple lemma.

Lemma 1. *Let T be the tree formed after the execution of some number of steps of the DJP algorithm. Let e and f be two arbitrary edges, each with exactly one endpoint in T, and let g be the maximum weight edge on the path from e to f in T. Then g cannot be heavier than both e and f.*

2.3 The Dense Case Algorithm

The procedure DenseCase$(G; F)$ takes as input an n_1-node, m_1-edge graph G, where $m_1 \leq m$ and $n_1 \leq n/\log^{(3)} n$, and returns the MSF F of G. It is not difficult to see that the algorithms presented in [FT87,GGST86,Chaz97,Chaz99], [Pet99] will find the MSF of G in $O(n + m)$ time.

2.4 Soft Heap

The main data structure used by our algorithm is the *Soft Heap* [Chaz98]. The Soft Heap is a kind of priority queue that gives us an optimal tradeoff between accuracy and speed. It supports the following operations:

- `MakeHeap`(): returns an empty soft heap.
- `Insert`(S, x): insert item x into heap S.
- `Findmin`(S): returns item with smallest key in heap S.
- `Delete`(S, x): delete x from heap S.
- `Meld`(S_1, S_2): create new heap containing the union of items stored in S_1 and S_2, destroying S_1 and S_2 in the process.

All operations take constant amortized time, except for Insert, which takes $O(\log(\frac{1}{\epsilon}))$ time. To save time the Soft Heap allows items to be grouped together and treated as though they have a single key. An item adopts the largest key of any item in its group, *corrupting* the item if its new key differs from its original key. Thus the original key of an item returned by Findmin (i.e. any item in the group with minimum key) is no more than the keys of all uncorrupted items in the heap. The guarantee is that after n Insert operations, no more than ϵn corrupted items are in the heap. The following result is shown in [Chaz98].

Lemma 2. *Fix any parameter $0 < \epsilon < 1/2$, and beginning with no prior data, consider a mixed sequence of operations that includes n inserts. On a Soft Heap the amortized complexity of each operation is constant, except for insert, which takes $O(\log(1/\epsilon))$ time. At most ϵn items are corrupted at any given time.*

3 A Key Lemma and Procedure

3.1 A Robust Contraction Lemma

It is well known that if T is a tree of MSF edges, we can *contract* T into a single vertex while maintaining the invariant that the MSF of the contracted graph plus T gives the MSF for the graph before contraction.

In our algorithm we will find a tree of MSF edges T in a *corrupted* graph, where some of the edge weights have been increased due to the use of a Soft Heap. In the lemma given below we show that useful information can be obtained by contracting certain corrupted trees, in particular those constructed using some number of steps from the Dijkstra-Jarnik-Prim (DJP) algorithm. Ideas similar to these are used in Chazelle's 1997 algorithm [Chaz97], and more explicitly in the recent algorithms of Pettie [Pet99] and Chazelle [Chaz99].

Before stating the lemma, we need some notation and preliminary concepts. Let $V(G)$ and $E(G)$ be the vertex and edge sets of G. Let the G-weight of an edge be its weight in graph G (the G may be omitted if implied from context).

For the following definitions, M and C are subgraphs of G. Denote by $G \Uparrow M$ a graph derived from G by raising the weight of each edge in M by arbitrary amounts (these edges are said to be corrupted). Let M_C be the set of edges in M with exactly one endpoint in C. Let $G \backslash C$ denote the graph obtained by contracting all connected components induced by C, i.e. by replacing each connected component with a single vertex and reassigning edge endpoints appropriately.

We define a subgraph C of G to be *DJP-contractible* if after executing the DJP algorithm on G for some number of steps, with a suitable start vertex in C, the tree that results is a spanning tree for C.

Lemma 3. *Let M be a set of edges in a graph G. If C is a subgraph of G that is DJP-contractible w.r.t. $G \Uparrow M$, then $MSF(G)$ is a subset of $MSF(C) \cup MSF(G \backslash C - M_C) \cup M_C$.*

Proof. Each edge in C that is not in MSF(C) is the heaviest edge on some cycle in C. Since that cycle exists in G as well, that edge is not in MSF(G). So we need only show that edges in $G \backslash C$ that are not in MSF($G \backslash C - M_C$) $\cup M_C$ are also not in MSF(G).

Let $H = G \backslash C - M_C$; hence we need to show that no edge in $H - MSF(H)$ is in $MSF(G)$. Let e be the heaviest edge on some cycle χ in H (i.e. $e \in H - MSF(H)$). If χ does not involve the vertex derived by contracting C, then it exists in G as well and $e \notin MSF(G)$. Otherwise, χ forms a *path* $\mathcal{P}$ in G whose end points, say x and y, are both in C. Let the end edges of $\mathcal{P}$ be (x, w) and (y, z). Since H included no corrupted edges with one end point in C, the G-weight of these edges is the same as their $(G \Uparrow M)$-weight.

Let T be the spanning tree of $C \Uparrow M$ derived by the DJP algorithm, $\mathcal{Q}$ be the path in T connecting x and y, and g be the heaviest edge in $\mathcal{Q}$. Notice that $\mathcal{P} \cup \mathcal{Q}$ forms a cycle. By our choice of e, it must be heavier than both (x, y) and (w, z), and by Lemma 1, the heavier of (x, y) and (w, z) is heavier than the $(G \Uparrow M)$-weight of g, which is an upper bound on the G-weights of all edges in $\mathcal{Q}$. So w.r.t. G-weights, e is the heaviest edge on the cycle $\mathcal{P} \cup \mathcal{Q}$ and cannot be in $MSF(G)$.

3.2 The Partition Procedure

Our algorithm uses the Partition procedure given below. This procedure finds DJP-contractible subgraphs $C_1, \ldots, C_k$ in which edges are progressively being corrupted by the Soft Heap. Let M_{C_i} contain only those corrupted edges with one endpoint in C_i at the time it is completed.

Each subgraph C_i will be DJP-contractible w.r.t a graph derived from G by several rounds of contractions and edge deletions. When C_i is finished it is contracted and all incident corrupted edges are discarded. By applying Lemma 3 repeatedly we see that after C_i is built, the MSF of G is a subset of

$$\bigcup_{j=1}^{i} MSF(C_j) \cup MSF\left(G \backslash \bigcup_{j=1}^{i} C_j - \bigcup_{j=1}^{i} M_{C_j}\right) \cup \bigcup_{j=1}^{i} M_{C_j}$$

Below, arguments appearing before the semicolon are inputs; the outputs will be returned in the other arguments. M is a set of edges and $\mathcal{C}=\{C_1, \ldots, C_k\}$ is a set of subgraphs of G. No edge will appear in more than one of $M, C_1, \ldots, C_k$.

Partition$(G, maxsize, \epsilon\,;\ M, \mathcal{C})$

```
All vertices are initially ``live''
M := ∅
i := 0
While there is a live vertex
    Increment i
    Let V_i := {v}, where v is any live vertex
    Create a Soft Heap consisting of v's edges (uses ε)
    While all vertices in V_i are live and |V_i| < maxsize
       Repeat
          Find and delete min-weight edge (x, y) from Soft Heap
       Until y ∉ V_i   (Assume w.l.o.g. x ∈ V_i)
       V_i := V_i ∪ {y}
       If y is live, insert each of y's edges into the Soft Heap
    Set all vertices in V_i to be dead
    Let M_{V_i} be the corrupted edges with one endpoint in V_i
    M := M ∪ M_{V_i};        G := G − M_{V_i}
    Dismantle the Soft Heap
Let C := {C_1, ..., C_i} where C_z is the subgraph induced by V_z
Exit.
```

Initially, Partition sets every vertex to be *live*. The objective is to convert each vertex to *dead*, signifying that it is part of a component C_i with $\leq maxsize$

vertices and part of a *conglomerate* of $\geq maxsize$ vertices, where a conglomerate is a connected component of the graph $\bigcup E(C_i)$. Intuitively a conglomerate is a collection of C_i's linked by common vertices. This scheme for growing components is similar to the one given in [FT87].

We grow the C_i's one at a time according to the DJP algorithm, except that we use a Soft Heap. A component is done growing if it reaches $maxsize$ vertices or if it attaches itself to an existing component. Clearly if a component does not reach $maxsize$ vertices, it has linked to a conglomerate of at least $maxsize$ vertices. Hence all its vertices can be designated dead. Upon completion of a component C_i, we discard the set of corrupted edges with one endpoint in C_i.

The running time of Partition is dominated by the heap operations, which depend on ϵ. Each edge is inserted into a Soft Heap no more than twice (once for each endpoint), and extracted no more than once. We can charge the cost of dismantling the heap to the insert operations which created it, hence the total running time is $O(m \log(\frac{1}{\epsilon}))$. The number of discarded edges is bounded by the number of insertions scaled by ϵ, thus $|M| \leq 2\epsilon m$. Thus we have

Lemma 4. *Given a graph G, any $0 < \epsilon < \frac{1}{2}$, and a parameter $maxsize$, Partition finds edge-disjoint subgraphs $M, C_1, \ldots, C_k$ in time $O(|E(G)| \cdot \log(\frac{1}{\epsilon}))$ while satisfying several conditions:*
a) For all $v \in V(G)$ there is some i s.t. $v \in V(C_i)$.
b) For all i, $|V(C_i)| \leq maxsize$.
c) For each conglomerate $P \in \bigcup_i C_i$, $|V(P)| \geq maxsize$.
d) $|E(M)| \leq 2\epsilon \cdot |E(G)|$
e) $MSF(G) \subseteq \bigcup_i MSF(C_i) \cup MSF(G \backslash (\bigcup_i C_i) - M) \cup M$

4 Overview of the Optimal Algorithm

Here is an overview of our optimal MSF algorithm.

- In the first stage we find DJP-contractible subgraphs $C_1, C_2, \ldots, C_k$ with their associated set of edges $M = \bigcup_i M_{C_i}$, where M_{C_i} consists of corrupted edges with one endpoint in C_i.
- In the second stage we find the MSF F_i of each C_i, and the MSF F_0 of the contracted graph $G \backslash (\bigcup_i C_i) - \bigcup_i M_{C_i}$. By Lemma 3, the MSF of the whole graph is contained within $F_0 \cup \bigcup_i (F_i \cup M_{C_i})$. Note that at this point we have not identified any edges as being in the MSF of the original graph G.
- In the third stage we find some MSF edges, via Borůvka steps, and recurse on the graph derived by contracting these edges.

We execute the first stage using the Partition procedure described in the previous section.

We execute the second stage with *optimal decision trees*. Essentially, these are hardwired algorithms designed to compute the MSF of a graph using an optimal number of edge-weight comparisons. In general, decision trees are much larger than the size of the problem that they solve and finding optimal ones is very time consuming. We can afford the cost of building decision trees by guaranteeing that each one is extremely small. At the same time, we make each

conglomerate formed by the C_i to be sufficiently large so that the MSF F_0 of the contracted graph can be found in linear time using the DenseCase algorithm.

Finally, in the third stage, we have a reduction in vertices due to the Borůvka steps, and a reduction in edges due to the application of Lemma 3. In our optimal algorithm both vertices and edges reduce by a constant factor, thus resulting in the recursive applications of the algorithm on graphs with geometrically decreasing sizes.

4.1 Decision Trees

An MSF decision tree is a rooted binary tree having an edge-weight comparison associated with each internal node (e.g. $weight(x, y) < weight(w, z)$). The left child represents that the comparison is true, the right child that it is false. Associated with each leaf is a spanning forest. An MSF decision tree is *correct* if the edge-weight comparisons encountered on any path from the root to a leaf uniquely identify the spanning forest at that leaf as the MSF. A decision tree is *optimal* if it is correct and there exists no correct decision tree with lesser depth.

Using brute force search, the optimal MSF decision trees for all graphs on $\leq \log^{(3)} n$ vertices may be constructed and checked in $o(n)$ time.

Our algorithm we will use a procedure DecisionTree($\mathcal{G}; \mathcal{F}$), which takes as input a collection of graphs $\mathcal{G}$, each with at most $\log^{(3)} n$ vertices, and returns their minimum spanning forests in $\mathcal{F}$ using the precomputed decision trees.

5 The Algorithm

As discussed above, the optimal MSF algorithm is as follows. First, precompute the optimal decision trees for all graphs with $\leq \log^{(3)} n$ vertices. Next, divide the input graph into subgraphs $C_1, C_2, ..., C_k$, discarding the set of corrupted edges M_{C_i} as each C_i is completed. Use the decision trees found earlier to compute the MSF F_i of each C_i, then contract each connected component spanned by $F_1 \cup \ldots \cup F_k$ (i.e., each conglomerate) into a single vertex. The resulting graph has $\leq n/\log^{(3)} n$ vertices since each conglomerate has at least $\log^{(3)} n$ vertices by Lemma 4. Hence we can use the DenseCase algorithm to compute its MSF F_0 in time linear in m. At this point, by Lemma 3 the MSF is now contained in the edge set $F_0 \cup \ldots \cup F_k \cup M_{C_1} \cup \ldots M_{C_k}$. On this graph we apply two Borůvka steps and then compute its MSF recursively. The algorithm is given below.

OptimalMSF(G)

```
If E(G) = ∅ then Return(∅)
Let ε := 1/8 and maxsize := log^(3) |V(G)|
Partition(G, maxsize, ε;  M, C)
DecisionTree(C;  F)
Let k := |C| and let C = {C_1, ..., C_k}, F = {F_1, ..., F_k}
G_a := G\(F_1 ∪ ... ∪ F_k) − M
DenseCase(G_a;  F_0)
G_b := F_0 ∪ F_1 ∪ ... ∪ F_k ∪ M
Boruvka2(G_b;  F', G_c)
F := OptimalMSF(G_c)
Return(F ∪ F')
```

Apart from recursive calls and using the decision trees, the computation performed by OptimalMSF is clearly linear since Partition takes $O(m \log(\frac{1}{\epsilon}))$ time, and owing to the reduction in vertices, the call to DenseCase also takes linear time. For $\epsilon = \frac{1}{8}$, the number of edges passed to the final recursive call is $\leq m/4 + n/4 \leq m/2$, giving a geometric reduction in the number of edges. Since no MSF algorithm can do better than linear time, the bottleneck, if any, must lie in using the decision trees, which are optimal by construction.

More concretely, let $T(m,n)$ be the running time of OptimalMSF. Let $\mathcal{T}^*(m,n)$ be the optimal number of comparisons needed on any graph with n vertices and m edges and let $\mathcal{T}^*(G)$ be the optimal number of comparisons needed on a *specific* graph G. The recurrence relation for T is given below. For the base case note that the graphs in the recursive calls will be connected if the input graph is connected. Hence the base case graph has no edges and one vertex, and we have $T(0,1)$ equal to a constant.

$$T(m,n) \leq \sum_i \mathcal{T}^*(C_i) + T(m/2, n/4) + c_1 \cdot m$$

It is straightforward to see that if $\mathcal{T}^*(m,n) = O(m)$ then the above recurrence gives $T(m,n) = O(m)$. One can also show that $T(m,n) = O(\mathcal{T}^*(m,n))$ for many natural functions for $\mathcal{T}^*$ (including $m \cdot \alpha(m,n)$). However, to show that this result holds no matter what the function describing $\mathcal{T}^*(m,n)$ is, we need to establish some results on the decision tree complexity of the MSF problem, which we do in the next section.

5.1 Some Results for MSF Decision Trees

In this section we establish some results on MSF decision trees that allow us to establish our main result that OptimalMSF runs in $O(\mathcal{T}^*(m,n))$ time.

Proving the following propositions is straightforward.

Proposition 1. $\mathcal{T}^*(m,n) \geq m/2$.

Proposition 2. *For fixed m and $n' > n$, $\mathcal{T}^*(m,n') \geq \mathcal{T}^*(m,n)$.*

Proposition 3. *For fixed n and $m' > m$, $\mathcal{T}^*(m',n) \geq \mathcal{T}^*(m,n)$.*

We now state a property that is used by Lemmas 5 and 6.

Property 1. The structure of G dictates that $\mathrm{MSF}(G) = \mathrm{MSF}(C_1) \cup \ldots \cup \mathrm{MSF}(C_k)$, where $C_1, \ldots, C_k$ are edge-disjoint subgraphs of G.

If $C_1, \ldots, C_k$ are the components returned by Partition, it can be seen that the graph $\bigcup_i C_i$ satisfies Definition 1 since every simple cycle in this graph must be contained in exactly one of the C_i.

The proof of the following lemma can be found in [PR99b].

Lemma 5. *If Property 1 holds for G, then there exists an optimal MSF decision tree for G which makes no comparisons of the form $e < f$ where $e \in C_i$, $f \in C_j$ and $i \neq j$.*

Lemma 6. *If Property 1 holds for G, then $\mathcal{T}^*(G) = \sum_i \mathcal{T}^*(C_i)$.*

Proof. (Sketch) Given an optimal decision tree T for G as in Lemma 5, we show that T can be transformed into a 'canonical' decision tree T' for G of the same height such that in T', comparisons for C_i precede comparisons for C_{i+1}, for each i, and further, the subgraph of T' containing the comparisons within C_i consists of isomorphic trees. This establishes the desired result since T' must contain a path that is the concatenation of the longest path in an optimal decision tree for each of the C_i. For details see [PR99b].

Corollary 1. *Let the C_i be the components formed by Partition when applied to G. Then $\sum_i \mathcal{T}^*(C_i) = \mathcal{T}^*(G) \leq \mathcal{T}^*(m,n)$.*

Corollary 2. *For any m and n, $2 \cdot \mathcal{T}^*(m,n) \leq \mathcal{T}^*(2m,2n)$*

We can now solve the recurrence relation given in the previous section.

$$\begin{aligned} T(m,n) &\leq \sum_i \mathcal{T}^*(C_i) + T(m/2,n/4) + c_1 \cdot m \\ &\leq \mathcal{T}^*(m,n) + T(m/2,n/4) + c_1 \cdot m \quad \textit{(Corollary 1)} \\ &\leq \mathcal{T}^*(m,n) + c \cdot \mathcal{T}^*(m/2,n/4) + c_1 \cdot m \quad \textit{(assume inductively)} \\ &\leq \mathcal{T}^*(m,n)(1 + c/2 + 2c_1) \quad \textit{(Corollary 2 and Propositions 1, 2)} \\ &\leq c \cdot \mathcal{T}^*(m,n) \ \textit{(for sufficiently large c; this completes the induction)} \end{aligned}$$

This gives us the desired theorem.

Theorem 1. *Let $\mathcal{T}^*(m,n)$ be the decision-tree complexity of the MSF problem on graphs with m edges and n nodes. Algorithm* `OptimalMSF` *computes the MSF of a graph with m edges and n vertices deterministically in $O(\mathcal{T}^*(m,n))$ time.*

6 Performance on Random Graphs

Even if we assume that MST has some super-linear complexity, we show below that our algorithm runs in linear time for nearly all graphs, regardless of edge weights. This improves upon the expected linear-time result of Karp and Tarjan [KT80], which depended on the edge weights being chosen randomly. Our result may also be contrasted with the randomized algorithm of Karger et al. [KKT95], which is shown to run in $O(m)$ time w.h.p. by a proof that depends on the permutation of edge weights and random bits chosen, not the graph topology. Throughout this section α will denote $\alpha(m,n)$. Most proofs in this section are omitted due to lack of space.

Theorem 2. *With probability $1 - e^{-\Omega(m/\alpha^2)}$, the MST of a graph drawn from $G_{n,m}$ can be found in linear time, regardless of the permutation of edge weights.*

In the next section we describe the *edge-addition martingale* for the $G_{n,m}$ model. In section 6.2 we use this martingale and Azuma's inequality to prove Theorem 2.

6.1 The Edge-Addition Martingale

We use the $G_{n,m}$ random graph model, that is, each graph with n labeled vertices and m edges is equally likely (the result can be extended to $G_{n,p}$). For analytical purposes, we select a random graph by beginning with n vertices and adding one edge at a time [ER61]. Let X_i be a random edge s.t. $X_i \neq X_j$ for $j < i$, and

$G_i = \{X_1, \ldots, X_i\}$ be the graph made up of the first i edges, with G_0 being the graph on n vertices having no edges.

We prove that if g is any graph-theoretic function and $g_E(G_i) = \mathrm{E}[g(G_m)|G_i]$, then $g_E(G_i)$, for $0 \le i \le m$ is a *martingale*. We call this the *edge-addition* martingale in contrast to the *edge-exposure* martingale for $G_{n,p}$.

A martingale is a sequence of random variables $Y_0, Y_1, \ldots, Y_m$ such that $\mathrm{E}[Y_i \,|\, Y_{i-1}] = Y_{i-1}$ for $0 < i \le m$.

Lemma 7. *The sequence $g_E(G_i) = \mathrm{E}[g(G_m) \,|\, G_i]$, for $0 \le i \le m$, is a martingale, where g is any graph theoretic function, G_0 is the edge-free graph on n vertices, and G_i is derived from G_{i-1} by adding a random edge not in G_{i-1} to G_{i-1}.*

We now recall the well-known Azuma's inequality (see, e.g., [AS92]).

Theorem 3. *(Azuma'a Inequality.) Let $Y_0, \cdots, Y_m$ be a martingale with $|Y_i - Y_{i-1}| \le 1$ for $0 < i \le m$. Let $\lambda > 0$ be arbitrary. Then $\Pr[|Y_m - Y_0| > \lambda\sqrt{m}] < e^{-\lambda^2/2}$.*

To facilitate the application of Azuma's inequality to the edge-addition martingale we give the following lemma.

Lemma 8. *Consider the sequence proved to be a martingale in Lemma 7. Let g be any graph-theoretic function such that $|g(G) - g(G')| \le 1$ for any pair of graphs G and G' of the form $G = H \cup \{e\}$ and $G' = H \cup \{e'\}$, for some graph H. Then $|g_E(G_i) - g_E(G_{i-1})| \le 1$, for $0 < i \le m$.*

6.2 Analysis

We define the *excess* of a subgraph H to be $|E(H)| - |F(H)|$, where $F(H)$ is any spanning forest of H. *Let $f(G)$ be the maximum excess of the graph made up of intra-component edges, where the sets of components range over all possible sets returned by the Partition procedure.* (Recall that the size of any component is no more than $k = maxsize = \log^{(3)} n$.)

Our key observation is that each pass of our optimal algorithm definitely runs in linear time if $f(G) \le m/\alpha(m,n)$. To see this, note that if this bound on $f(G)$ holds, we can reduce the *total* number of intra-component edges to $\le 2m/\alpha$ in linear time using $\log \alpha$ Borůvka steps, and then, clearly, the MST of the resulting graph can be determined in $O(m)$ time. We show below that if a graph is randomly chosen from $G_{n,m}$, $f(G) \le m/\alpha(m,n)$ with high probability.

Define $f_E(G_i) = \mathrm{E}[f(G_m)|G_i]$. The following lemma gives a bound on $f_E(G_0)$; its proof is straightforward.

Lemma 9. $f_E(G_0) = o(m/\alpha)$.

The following two lemmas establish the application of Azuma's inequality to the graph-theoretic function f.

Lemma 10. *Let $G = H \cup \{e\}$ and $G' = H \cup \{e'\}$ be two graphs on a set of labeled vertices which differ by no more than one edge. Then $|f(G) - f(G')| \le 1$.*

Lemma 11. *Let G be chosen from $G_{n,m}$. Then $\Pr[f(G) > m/\alpha] < e^{-\Omega(m/\alpha^2)}$.*

Proof. (of **Theorem 2**.) We examine only the first $\log k$ passes of our optimal algorithm, since all remaining passes certainly take $o(m)$ time. Lemma 11 assures us that the first pass runs in linear time w.h.p. However, the topology of the graph examined in later passes *does* depend on the edge weights. Assuming the Borůvka steps contract all parts of the graph at a constant rate, which can easily be enforced, a partition of the graph in one pass of the algorithm corresponds to a partition of the original graph into components of size less than k^c, for some fixed c. Using k^c in place of k does not affect Lemma 9, which gives the Theorem.

7 Discussion

An intriguing aspect of our algorithm is that we do not know its precise deterministic running time although we can prove that it is within a constant factor of optimal. Results of this nature have been obtained in the past for sensitivity analysis of minimum spanning trees [DRT92] and convex matrix searching [Lar90]. Also, for the problem of triangulating a convex polygon, it was observed in [DRT92] that an alternate linear-time algorithm could be obtained using optimal decision trees on small subproblems. However, these earlier algorithms make use of decision trees in more straightforward ways than the algorithm presented here.

As noted earlier, the construction of optimal decision trees takes sub-linear time. Thus, it is important to observe that the use of decision trees in our algorithm does not result in a large constant factor in the running time, nor does it result in an algorithm that is non-uniform.

It should be noted that the existence of a linear-time verification algorithm for MST immediately implies a naive optimal MST algorithm that is obtained by enumerating all possible algorithms, evaluating them incrementally, and verifying the outputs until we encounter the correct output. However, the constant factor for this algorithm is astronomical, and it sheds no light on the relationship between the algorithmic and decision-tree complexities of the problem.

8 Conclusion

We have presented a deterministic MSF algorithm that is provably optimal. The algorithm runs on a pointer machine, and on graphs with n vertices and m edges, its running time is $O(\mathcal{T}^*(m,n))$, where $\mathcal{T}^*(m,n)$ is the decision-tree complexity of the MSF problem on n-node, m-edge graphs. Also, on random graphs our algorithm runs in linear time with high probability for all possible edge-weights. Although the exact running time of our algorithm is not known, we have shown that the time bound depends only on the number of edge-weight comparisons needed to determine the MSF, and not on data structural issues.

Pinning down the function that describes the worst-case complexity of our algorithm is the main open question that remains for the sequential complexity of the MSF problem. A related question is the parallel work-time complexity of this problem. In this context, resolved recently were the randomized work-time complexity [PR99] and the deterministic time complexity [CHL99] of the MSF problem on the EREW PRAM. An open question that remains here is to obtain a deterministic work-time optimal parallel MSF algorithm.

References

AS92. N. Alon, J. Spencer. *The Probabilistic Method.* Wiley, New York, 1992.

Bor26. O. Borůvka . O jistém problému minimaálním. *Moravské Přírodovědecké Společnosti* 3, pp. 37-58, 1926. (In Czech).

Chaz97. B. Chazelle. A faster deterministic algorithm for minimum spanning trees. In *FOCS '97*, pp. 22–31, 1997.

Chaz98. B. Chazelle. Car-pooling as a data structuring device: The soft heap. In *ESA '98* (Venice), pp. 35–42, LNCS 1461, Springer, 1998.

Chaz99. B. Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. NECI Technical Report 99-099, 1999.

CHL99. K. W. Chong, Y. Han and T. W. Lam. On the parallel time complexity of undirected connectivity and minimum spanning trees. In *Proc. SODA*, pp. 225-234, 1999.

Dij59. E. W. Dijkstra. A note on two problems in connexion with graphs. In *Numer. Math.*, 1, pp. 269-271, 1959.

DRT92. B. Dixon, M. Rauch, R. E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Jour. Comput.*, vol 21, pp. 1184-1192, 1992.

ER61. P. Erdös, A. Rényi On the evolution of random graphs. *Bull. Inst. Internat. Statist.* 38, pp. 343–347, 1961.

FT87. M. L. Fredman, R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. In *JACM* 34, pp. 596–615, 1987.

FW90. M. Fredman, D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. In *Proc. FOCS '90*, pp. 719–725, 1990.

GGST86. H. N. Gabow, Z. Galil, T. Spencer, R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. In *Combinatorica* 6, pp. 109–122, 1986.

GH85. R. L. Graham, P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing* 7, pp. 43–57, 1985.

Jar30. V. Jarník. O jistém problému minimaálním. *Moravské Přírodovědecké Společnosti* 6, pp. 57-63, 1930. (In Czech).

KKT95. D. R. Karger, P. N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *JACM*, 42:321–328, 1995.

KT80. R. M. Karp, R. E. Tarjan. Linear expected-time algorithms for connectivity problems. *J. Algorithms* 1 (1980), no. 4, pp. 374–393.

Lar90. L. L. Larmore. An optimal algorithm with unknown time complexity for convex matrix searching. *IPL*, vol. 36, pp. 147-151, 1990.

PR99. S. Pettie, V. Ramachandran. A randomized time-work optimal parallel algorithm for finding a minimum spanning forest *Proc. RANDOM '99*, LNCS 1671, Springer, pp. 233-244, 1999.

PR99b. S. Pettie, V. Ramachandran. An optimal minimum spanning tree algorithm. Tech Report TR99-17, Univ. of Texas at Austin, 1999.

Pet99. S. Pettie. Finding minimum spanning trees in $O(m\alpha(m,n))$ time. Tech Report TR99-23, Univ. of Texas at Austin, 1999.

Prim57. R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389-1401, 1957.

Tar79. R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. In *JCSS*, 18(2), pp 110–127, 1979.

Improved Shortest Paths on the Word RAM

Torben Hagerup

Fachbereich Informatik
Johann Wolfgang Goethe-Universität Frankfurt
D–60054 Frankfurt am Main, Germany
hagerup@informatik.uni-frankfurt.de

Abstract. Thorup recently showed that single-source shortest-paths problems in undirected networks with n vertices, m edges, and edge weights drawn from $\{0, \ldots, 2^w - 1\}$ can be solved in $O(n + m)$ time and space on a unit-cost random-access machine with a word length of w bits. His algorithm works by traversing a so-called *component tree*. Two new related results are provided here. First, and most importantly, Thorup's approach is generalized from undirected to directed networks. The resulting time bound, $O(n + m \log w)$, is the best deterministic linear-space bound known for sparse networks unless w is superpolynomial in $\log n$. As an application, all-pairs shortest-paths problems in directed networks with n vertices, m edges, and edge weights in $\{-2^w, \ldots, 2^w\}$ can be solved in $O(nm + n^2 \log\log n)$ time and $O(n + m)$ space (not counting the output space). Second, it is shown that the component tree for an undirected network can be constructed in deterministic linear time and space with a simple algorithm, to be contrasted with a complicated and impractical solution suggested by Thorup. Another contribution of the present paper is a greatly simplified view of the principles underlying algorithms based on component trees.

1 Introduction

The *single-source shortest-paths* (*SSSP*) problem asks, given a network $\mathcal{N}$ with real-valued edge lengths and a distinguished vertex s in $\mathcal{N}$ called the *source*, for shortest paths in $\mathcal{N}$ from s to all vertices in $\mathcal{N}$ for which such shortest paths exist. It is one of the most fundamental and important network problems from both a theoretical and a practical point of view. Actually, the more fundamental and important problem is that of finding a shortest path from s to a single given vertex t, but this does not appear to be significantly easier than solving the complete SSSP problem with source s.

This paper considers mainly the important special case of the SSSP problem in which all edge lengths are nonnegative. The classic algorithm for this special case is *Dijkstra's algorithm* [5,6,24]. Dijkstra's algorithm maintains for every vertex v in $\mathcal{N}$ a *tentative distance* from s to v, processes the vertices one by one, and always selects as the next vertex to be processed one whose tentative distance is minimal. The operations that need to be carried out on

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 61–72, 2000.
© Springer-Verlag Berlin Heidelberg 2000

the set of unprocessed vertices—in particular, identifying a vertex with minimal *key* (= tentative distance)—are supported by the *priority-queue* data type, and therefore an efficient implementation of Dijkstra's algorithm essentially boils down to an efficient realization of the priority queue.

Suppose that the graph underlying $\mathcal{N}$ is $G = (V, E)$ and take $n = |V|$ and $m = |E|$. An implementation of Dijkstra's algorithm with a running time of $O(n + m \log n)$ is obtained by realizing the priority queue through a binary heap or a balanced search tree. Realizing the priority queue by means of a *Fibonacci heap*, Fredman and Tarjan [8] lowered the time bound to $O(m + n \log n)$. The priority queues mentioned so far are comparison-based. In reality, however, edge lengths are numeric quantities, and very frequently, they are or can be viewed as integers. In the remainder of the paper, we make this assumption, which lets a host of other algorithms come into play.

Our model of computation is the *word RAM*, which is like the classic unit-cost random-access machine, except that for an integer parameter $w \geq 1$ called the *word length*, the contents of all memory cells are integers in the range $\{0, \ldots, 2^w - 1\}$, and that some additional instructions are available. Specifically, the available unit-time operations are assumed to include addition and subtraction, (noncyclic) bit shifts by an arbitrary number of positions, and bitwise boolean operations, but not multiplication (the *restricted instruction set*). Our algorithm for undirected networks in addition assumes the availability of a unit-time "most-significant-bit" (MSB) instruction that, applied to a positive integer r, returns $\lfloor \log r \rfloor$ (all logarithms are to base 2). When considering an instance of the SSSP problem with n vertices, we assume that $w > \log n$, since otherwise n is not a representable number. In the same vein, when nothing else is stated, edge weights are assumed to be integers in the range $\{0, \ldots, 2^w - 1\}$.

We now discuss previous algorithms for the SSSP problem that work on the word RAM, focusing first on deterministic algorithms. A well-known data structure of van Emde Boas et al. [23] is a priority queue that allows insertion and deletion of elements with keys in $\{0, \ldots, C\}$ as well as the determination of an element with minimal key in $O(\log \log C)$ time per operation. This implies an SSSP algorithm with a running time of $O(n + m \log w)$. In more recent work, Thorup [18] improved this to $O(n + m \log \log n)$, the best bound known for sparse networks. Both algorithms, however, use $2^{\Theta(w)}$ space, which makes them impractical if w is larger than $\log n$ by a nonnegligible factor. A different algorithm by Thorup [20] achieves $O(n + m(\log \log n)^2)$ time using linear space, $O(n + m)$. Algorithms that are faster for denser networks were indicated by Ahuja et al. [2], Cherkassky et al. [4], and Raman [14,15,16]; their running times are of the forms $O(m + n(\log n)^{\Theta(1)})$ and $O(m + nw^{\Theta(1)})$. Some of these algorithms employ randomization, multiplication, and/or superlinear space. Using randomization, an expected running time of $O(n + m \log \log n)$ can be achieved in linear space [19].

Our first result is a new deterministic algorithm for the SSSP problem that works in $O(n + m \log w)$ time. The time bound is never better than that of Thorup [18]. The new algorithm, however, works in linear space. For sparse networks, the new algorithm is faster than all previous deterministic linear-

space solutions unless w is superpolynomial in $\log n$. We actually prove a more general result: If the edges can be partitioned into $b \geq 2$ groups such that the lengths of the edges within each group differ by at most a constant factor, the SSSP problem can be solved in $O(n \log\log n + m \log b)$ time. Our construction implies that the all-pairs shortest-paths (*APSP*) problem in networks with n vertices, m edges, and edge weights drawn from $\{-2^w, \ldots, 2^w\}$ can be solved in $O(nm + n^2 \log\log n)$ time and $O(n + m)$ space (not counting the output space). No faster APSP algorithm is known for any combination of n and m. If $m = \omega(n)$ and $m = o(n \log n)$, the new algorithm is faster than all previous algorithms.

In a remarkable development, Thorup [21] showed that the SSSP problem can be solved in linear time and space for undirected networks. His algorithm is all the more interesting in that it is not a new implementation of Dijkstra's algorithm. The vertices are still processed one by one, but the strict processing in the order of increasing tentative distance is abandoned in favor of a more permissive regime, the computation being structured with the aid of a so-called *component tree*. Thorup provides two algorithms for constructing the component tree. One uses the *Q-heap* data structure of Fredman and Willard [9] and works in $O(n + m)$ time, but is complicated and utterly impractical. The other one is simple and conceivably practical, but its running time is $O(m\alpha(m, n))$, where α is an "inverse Ackermann" function known from the analysis of a union-find data structure [17]. Our second result is a procedure for computing the component tree of an undirected network that is about as simple as Thorup's second algorithm, but works in linear time and space.

2 Shortest Paths in Directed Networks

This section proves our main result:

Theorem 1. *For all positive integers n, m and w with $w > \log n \geq 1$, single-source shortest-paths problems in networks with n vertices, m edges, and edge lengths in the range $\{0, \ldots, 2^w - 1\}$ can be solved in $O(n + m \log w)$ time and $O(n + m)$ space on a word RAM with a word length of w bits and the restricted instruction set.*

Let us fix a network $\mathcal{N}$ consisting of a directed graph $G = (V, E)$ and a length function $c : E \to \{0, \ldots, 2^w - 1\}$ as well as a source $s \in V$ and take $n = |V| \geq 2$ and $m = |E|$. We assume without loss of generality that G is strongly connected, i.e., that every vertex is reachable from every other vertex. Then $m \geq n$, and we can define $\delta(v)$ as the length of a shortest path in G from s to v, for all $v \in V$. It is well-known that knowledge of $\delta(v)$ for all $v \in V$ allows us, in $O(m)$ time, to compute a shortest-path tree of G rooted at s (see, e.g., [1, Section 4.3]), so our task is to compute $\delta(v)$ for all $v \in V$.

Dijkstra's algorithm for computing $\delta(v)$ for all $v \in V$ can be viewed as simulating a fire starting at s at time 0 and propagating along all edges at unit speed. The algorithm maintains for each vertex v an upper bound $d[v]$ on the (simulated) time $\delta(v)$ when v will be reached by the fire, equal to the time when

v will be reached by the fire from a vertex u already on fire with $(u, v) \in E$ (∞ if there is no such vertex). Whenever a vertex u is attained by the fire (u is *visited*), the algorithm reconsiders $d[v]$ for each unvisited vertex v with $(u, v) \in E$ and, if the current estimate $d[v]$ is larger than the time $d[u] + c(u, v)$ when v will be hit by the fire from u, decreases $d[v]$ to $d[u] + c(u, v)$; in this case we say that the edge (u, v) is *relaxed*. The simulated time is then stepped to the time of the next visit of a vertex, which is easily shown to be the minimal d value of an unvisited vertex.

Consider a distributed implementation of Dijkstra's algorithm in which each vertex v is simulated by a different processor P_v. The relaxation of an edge (u, v) is implemented through a message sent from P_u to P_v and specifying the new upper bound on $\delta(v)$. For each $v \in V$, P_v receives and processes all messages pertaining to relaxations of edges into v, then reaches the simulated time $d[v]$ and visits v, and subsequently sends out an appropriate message for each edge out of v that it relaxes. The implementation remains correct even if the processors do not agree on the simulated time, provided that each message is received in time: For each vertex v, a message specifying an upper bound of t on $\delta(v)$ should be received by P_v before it advances its simulated time beyond t. If such a message corresponds to the relaxation of an edge $e = (u, v)$, it was generated and sent by P_u at its simulated time $t - c(e)$. Provided that messages have zero transit times, this shows that for all $e = (u, v) \in E$, we can allow the simulated time of P_u to lag behind that of P_v by as much as $c(e)$ without jeopardizing the correctness of the implementation. In order to capitalize on this observation, we define a *component tree* as follows.

Take the *level* of each edge $e \in E$ to be the integer i with $2^{i-1} \leq c(e) < 2^i$ if $c(e) > 0$, and 0 if $c(e) = 0$. For each integer i, let G_i be the subgraph of G spanned by the edges of level at most i. A *component tree* for $\mathcal{N}$ is a tree T, each of whose nodes x is marked with a *level* in the range $\{-1, \ldots, w\}$, $level(x)$, and a *priority* in the range $\{0, \ldots, n-1\}$, $priority(x)$, such that the following conditions hold:

1. The leaves of T are exactly the vertices in G, and every inner node in T has at least two children.
2. Let x and y be nodes in T, with x the parent of y. Then $level(x) > level(y)$.
3. Let u and v be leaf descendants of a node x in T. Then there is a path from u to v in $G_{level(x)}$.
4. Let u and v be leaf descendants of distinct children y and z, respectively, of a node x in T. Then $priority(y) < priority(z)$ or there is no path from u to v in $G_{level(x)-1}$.

The component tree is a generalization of the component tree of Thorup [21]. Let $T = (V_T, E_T)$ be a component tree for $\mathcal{N}$ and, for all $x \in V_T$, let G_x be the subgraph of $G_{level(x)}$ spanned by the leaf descendants of x. The conditions imposed above can be shown to imply that for every $x \in V_T$, G_x is a strongly connected component (SCC) of $G_{level(x)}$, i.e., a maximal strongly connected subgraph of $G_{level(x)}$.

We carry out a simulation of the distributed implementation discussed above, imagining the processor P_v as located at the leaf v of T, for each $v \in V$. The rest of T serves only to enforce a limited synchronization between the leaf processors, as follows: For each integer i, define an *i-interval* to be an interval of the form $[r \cdot 2^i, (r+1) \cdot 2^i)$ for some integer $r \geq 0$. Conceptually, the algorithm manipulates *tokens*, where an i-token is an abstract object labeled with an i-interval, for each integer i, and a token is an i-token for some i. A nonroot node x in T occasionally receives from its parent a token labeled with an interval I, interpreted as a permission to advance its simulated time across I. If x has level i and is not a leaf, it then splits I into consecutive $(i-1)$-intervals $I_1, \ldots, I_k$ and, for $j = 1, \ldots, k$, steps through its children in an order of nondecreasing priorities and, for each child y, sends an $(i-1)$-token labeled with I_j to y and waits for a completion signal from y before stepping to its next child or to the next value of j. Once the last child of x has sent a completion signal for the last token to x, x sends a completion signal to its parent.

The root of T behaves in the same way, except that it neither generates completion signals nor receives tokens from a parent; we can pretend that the root initially receives a token labeled with the interval $[0, \infty)$. A leaf node v, upon receiving a token labeled with an interval I from its parent, checks whether $d[v] \in I$ and, if so, visits v and relaxes all edges leaving v that yield a smaller tentative distance. No "relaxation messages" need actually be generated; instead, the corresponding decreases of d values are executed directly. Similarly, although the simulation algorithm was described above as though each node in T has its own processor, it is easily turned into a recursive or iterative algorithm for a single processor.

Consider a relaxation of an edge $(u, v) \in E$ and let x, y, and z be as in condition (4) in the definition of a component tree. Then either $\mathit{priority}(y) < \mathit{priority}(z)$, in which case the simulated time of P_u never lags behind that of P_v, or $c(u, v) \geq \lfloor 2^{i-1} \rfloor$, where $i = \mathit{level}(x)$. Since the synchronization enforced by x never allows the simulated times of two processors at leaves in its subtree to differ by more than 2^{i-1}, our earlier considerations imply that the simulation is correct, i.e., the value of $\delta(v)$ is computed correctly for all $v \in V$.

As described so far, however, the simulation is not efficient. It is crucial not to feed tokens into a node x in T before the first token that actually enables a leaf descendant of x to be visited, and also to stop feeding tokens into x after the last leaf descendant of x has been visited. Thus each node x of T initially is *dormant*, then it becomes *active*, and finally it becomes *exhausted* (except for the root of T, which is always active). The transition of x from the dormant to the active state is triggered by the parent of x producing a token labeled with an interval that contains $d[x]$, defined to be the smallest d value of a leaf descendant of x. When the last leaf descendant of x has been visited, on the other hand, x notifies its parent that it wishes to receive no more tokens and enters the exhausted state. If x is an inner node, this simply means that x becomes exhausted when its last child becomes exhausted.

The following argument of Thorup [21] shows the total number of tokens exchanged to be $O(m)$: The number of tokens "consumed" by a node x in T of level i is at most 1 plus the ratio of the diameter of G_x to 2^i. The "contribution" of a fixed edge in E to the latter ratio, at various nodes x on a path in T, is bounded by $\sum_{j=0}^{\infty} 2^{-j} = 2$. Since T has fewer than $2n$ nodes, the total number of tokens is bounded by $2n + 2m$. In order to supply its children with tokens in the right order without violating the constraints implied by the children's priorities, each inner node in T initially sorts its children by their priorities. Using two-pass radix sort, this can be done together for all inner nodes in $O(n)$ total time. Taking the resulting sequence as the universe, each inner node subsequently maintains the set of its active children in a sorted list and, additionally, in a van Emde Boas tree [23]. The sorted list allows the algorithm to step to the next active child in constant time, and the van Emde Boas tree allows it to insert or delete an active child in $O(\log\log n)$ time. As there are $O(n)$ insertions and deletions of active nodes over the whole simulation, the total time needed is $O(m + n\log\log n) = O(m\log w)$. Since it is not difficult to implement the van Emde Boas tree in space proportional to the size of the universe [22], the total space needed by all instances of the data structure is $O(n)$.

If we ignore the time spent in constructing T and in discovering nodes that need to be moved from the dormant to the active state, the running time of the algorithm is dominated by the contribution of $O(m\log w)$ identified above. We now consider the two remaining problems.

2.1 Constructing the Component Tree

We show how to construct the component tree T in $O(m\min\{n, \log w\})$ time, first describing a simple, but inefficient algorithm.

The algorithm maintains a forest F that gradually evolves into the component tree. Initially $F = (V, \emptyset)$, i.e., F consists of n isolated nodes. Starting from a network $\mathcal{N}_{-1}$ that also contains the elements of V as isolated vertices and no edges, the algorithm executes $w+1$ *stages*. In Stage j, for $j = 0, \ldots, w$, a network $\mathcal{N}_j$ is obtained from $\mathcal{N}_{j-1}$ by inserting the edges in $\mathcal{N}$ of level j, computing the SCCs of the resulting network, and contracting the vertices of each nontrivial SCC to a single vertex. In F, each contraction of the vertices in a set U is mirrored by creating a new node that represents U, giving it level j, and making it the parent of each node in U. Suitable priorities for the vertices in U are obtained from a topological sorting of the (acyclic) subgraph of $\mathcal{N}_{j-1}$ spanned by the vertices in U. So that the remaining edges can be inserted correctly later, their endpoints are updated to reflect the vertex contractions carried out in Stage j. The resulting tree is easily seen to be a component tree.

Assuming that $w \leq m$, we lower the construction time from $O(mw)$ to $O(m\log w)$ by carrying out a preprocessing step that allows each stage to be executed with only the edges essential to that stage. For each $e = (u, v) \in E$, define the *essential level* of e to be the unique integer $i \in \{-1, \ldots, w-1\}$ such that u and v belong to distinct SCCs in G_i, but not in G_{i+1}. Starting with $E_{-1} = E$ and $E_0 = E_1 = \cdots = E_{w-1} = \emptyset$, the following recursive algorithm,

which can be viewed as a batched binary search, stores in E_i the subset of the edges in E of essential level i, for $i = -1, \ldots, w-1$. The outermost call is *BatchedSearch*$(-1, w)$.

procedure *BatchedSearch*(i, k):
  if $k - i \geq 2$ **then**
    $j := \lfloor (i+k)/2 \rfloor$;
    Let $\mathcal{N}_j$ be the subnetwork of $\mathcal{N}$ spanned by the edges in E_i of level $\leq j$;
    Compute the SCCs of $\mathcal{N}_j$;
    Move from E_i to E_j each edge with endpoints in distinct SCCs of $\mathcal{N}_j$;
    Contract each SCC of $\mathcal{N}_j$ to a single vertex
      and rename the endpoints of the edges in E_j accordingly;
    BatchedSearch(i, j);
    BatchedSearch(j, k);

The calls of *BatchedSearch* form a complete binary call tree of depth $O(\log w)$. If a call *BatchedSearch*(i, k) is associated with its lower argument i, each edge can be seen to belong to only one set E_i whose index i is associated with a call at a fixed level in the call tree. Since all costs of a call *BatchedSearch*(i, k), exclusive of those of recursive calls, are $O(1 + |E_i|)$, the execution time of the algorithm is $O(w + m \log w) = O(m \log w)$. Moreover, it can be seen that at the beginning of each call *BatchedSearch*(i, k), E_i contains exactly those edges in E whose endpoints belong to distinct SCCs in G_i, but not in G_k. Applied to the leaves of the call tree, this show the output of *BatchedSearch* to be as claimed.

We now use the original, simple algorithm, with the following modifications: (1) Instead of renaming edge endpoints explicitly, we use an efficient union-find data structure to map the endpoints of an edge to the nodes that resulted from them through a sequence of node contractions. Over the whole construction, the time needed for this is $O(m\alpha(m, n)) = O(m + n \log\log n) = O(m \log w)$ [17]. (2) In Stage j, for $j = 0, \ldots, w$, $\mathcal{N}_j$ is obtained from $\mathcal{N}_{j-1}$ by inserting each edge in $\mathcal{N}$ that was not inserted in an earlier stage, whose endpoints were not contracted into a common node, and whose level and essential level are both at most j. By the definition of the essential level of an edge, each edge disappears through a contraction no later than in the stage following its insertion, so that the total cost of the algorithm, exclusive of that of the union-find data structure, comes to $O(m)$. On the other hand, although the insertion of an edge may be delayed relative to the original algorithm, every edge is present when it is needed, so that the modified algorithm is correct.

We now sketch how to construct the component tree in $O(nm)$ time when $w \geq n$. Again, the basic approach is as in the simple algorithm. Starting with a graph that contains the elements of V as vertices and no edges, we insert the edges in E in an order of nondecreasing levels into a graph H, keeping track of the transitive closure of H as we do so. The transitive closure is represented through the rows and columns of its adjacency matrix, each of which is stored in a single word as a bit vector of length n. It is easy to see that with this representation, the transitive closure can be maintained in $O(n)$ time per edge insertion. After

each insertion of all edges of a common level, we pause to compute the strongly connected components, contract each of these and insert a corresponding node in a partially constructed component tree. This can easily be done in $O(kn)$ time, where k is the number of vertices taking part in a contraction. Over the whole computation, the time sums to $O(nm)$.

2.2 Activating the Nodes in the Component Tree

In order to be activated at the proper time, each nonroot node y in T needs to place a "wakeup request" with its parent x. To this effect, each active node x in T, on level i, say, is equipped with a *calendar* containing a slot for each $(i-1)$-interval of simulated time during which x is active. The calendar of x is represented simply as an array of (pointers to) linked lists, each list containing entries of all children of x requesting to be woken up at the corresponding $(i-1)$-interval. Since the total number of tokens exchanged is $O(m)$, calendars of total size $O(m)$ suffice, and the calendar of a node x can be allocated when x becomes active.

The wakeup mechanism requires us to maintain $d[y]$ for each dormant node y in T with an active parent x; let us call such a node *pre-active*. We describe below how to compute $d[y]$ at the moment at which y becomes pre-active. Subsequent changes to $d[y]$, up to the point at which y becomes active, are handled as follows: Whenever $d[v]$ decreases for some vertex $v \in V$, we locate the single pre-active ancestor y of v (how to do this is also described below) and, if appropriate, move the entry of y in the calendar of its parent to a different slot (in more detail, the entry is deleted from one linked list and inserted in another).

We list the leaves in T from left to right, calling them *points*, and associate each node in T with the interval consisting of its leaf descendants. When a node becomes pre-active, it notifies the last point v in its interval of this fact—we will say that v becomes a *leader*. Now the pre-active ancestor of a point can be determined by finding the leader of the point, the nearest successor of the point that is a leader. In order to do this, we divide the points into *intervals* of w points each and maintain for each interval a bit vector representing the set of leaders in the interval. Moreover, we keep the last point of each interval permanently informed of its current leader. Since T is of depth $O(w)$ and the number of intervals is $O(n/w)$, this can be done in $O(n)$ overall time, and now each point can find its current leader in $O(\log w)$ time.

In order to compute $d[y]$ when y becomes pre-active, we augment the data structure described so far with a complete binary tree planted over each interval and maintain for each node in the tree the minimum d value over its leaf descendants. Decreases of d values are easily executed in $O(\log w)$ time, updating along the path from the relevant leaf to the root of its tree. When a segment of length r is split, we compute the minima over each of the new segments by following paths from a leaf to the root in the trees in which the segment begins and ends and inspecting the roots of all trees in between, which takes $O(\log w + r/w)$ time. Since there are at most m decreases and $n-1$ segment splits and the total

length of all segments split is $O(nw)$, the total time comes to $O(m \log w)$. This ends the proof of Theorem 1.

2.3 Extensions

If only $b \geq 2$ of the $w+1$ possible edge levels $0, \ldots, w$ occur in an input network $\mathcal{N}$ with n vertices and m nodes, we can solve SSSP problems in $\mathcal{N}$ in $O(n \log\log n + m \log b)$ time and $O(n+m)$ space. For this, the "search space" of the algorithm *BatchedSearch* should be taken to be the actual set of edge levels (plus, possibly, one additional level needed to ensure strong connectivity), and the activation of nodes in the component tree should use intervals of size b rather than w. This changes all bounds of $O(m \log w)$ in the analysis to $O(m \log b)$, and all other time bounds are $O(m + n \log\log n)$. This also takes care of the case $w > m$ that was ignored in Section 2.1.

As observed by Johnson [12], the APSP problem in a strongly connected network $\mathcal{N}$ with n vertices, m edges, edge lengths in $\{-2^w, \ldots, 2^w\}$, and no negative cycles can be solved with an SSSP computation in $\mathcal{N}$ and $n-1$ SSSP computations in an auxiliary network $\mathcal{N}'$ with n vertices, m edges, and edge lengths in $\{0, \ldots, n2^w\}$. The SSSP computation in $\mathcal{N}$ can be carried out in $O(nm)$ time with the Bellman-Ford algorithm [3,7]. The SSSP computations in $\mathcal{N}'$ can be performed with the new algorithm, but constructing the component tree for $\mathcal{N}'$ only once. Disregarding the construction of the component tree and the activation of its nodes, the new algorithm works in $O(m + n \log\log n)$ time. The node activation can be done within the same time bound by appealing to a decrease-split-minimum data structure due to Gabow [10]; this connection was noted by Thorup [21], who provides details. Since the component tree can be constructed in $O(nm)$ time, this proves the following theorem.

Theorem 2. *For all positive integers n, m and w with $w > \log n \geq 1$, all-pairs shortest-paths problems in networks with n vertices, m edges, edge lengths in the range $\{-2^w, \ldots, 2^w\}$, and no negative cycles can be solved in $O(nm + n^2 \log\log n)$ time and $O(n+m)$ space (not counting the output space) on a word RAM with a word length of w bits and the restricted instruction set.*

3 Shortest Paths in Undirected Networks

When the algorithm of the previous section is applied to an undirected network, it is possible to eliminate the bottlenecks responsible for the superlinear running time. As argued by Thorup [21], the node activation can be done in $O(n+m)$ overall time by combining the decrease-split-minimum data structure of Gabow [10] with the Q-heap of Fredman and Willard [9]. The second bottleneck is the construction of the component tree, for which we propose a new algorithm.

3.1 The Component Tree for Undirected Networks

In the interest of simplicity, we will assume that there are no edges of zero length. This is no restriction, as all connected components of G_0 can be replaced by single vertices in a preprocessing step that takes $O(n+m)$ time.

We begin by describing a simple data structure that consists of a bit vector indexed by the integers $1, \ldots, w$ and an array, also indexed by $1, \ldots, w$, of stacks of edges. The bit vector typically indicates which of the stacks are nonempty. In constant time, we can perform a push or pop on a given stack and update the bit vector accordingly. Using the MSB instruction, we can also determine the largest index, smaller than a given value, of a nonempty stack or, by keeping the reverse bit vector as well, the smallest index, larger than a given value, of a nonempty stack. In particular, treating the stacks simply as sets, we can implement what [11] calls a *neighbor dictionary* for storing edges with their levels as keys that executes each operation in constant time. Since only the level of a key is relevant to the component tree, in the remainder of the section we will assume that the length of an edge is replaced by its level and denote the resulting network by $\mathcal{N}'$.

Following Thorup [21], we begin by constructing a minimum spanning tree (MST) of $\mathcal{N}'$. Instead of appealing to the MST algorithm of Fredman and Willard [9], however, we simply use Prim's MST algorithm [6,13], which maintains a subtree T of $\mathcal{N}'$, initially consisting of a single node, processes the edges in $\mathcal{N}'$ one by one, and always chooses the next edge to process as a shortest edge with at least one endpoint in T. Aided by an instance of the dictionary described above, we can execute Prim's algorithm to obtain an MST T_{M} of $\mathcal{N}'$ in $O(m)$ time. We root T_{M} at an arbitrary node. The significance of T_{M} is that a component tree for T_{M} (with the original edge lengths) is also a component tree for $\mathcal{N}$.

The next step is to perform a depth-first search of T_{M} with the aim of outputting a list of the edges of T_{M}, divided into *groups*. Whenever the search retreats over an edge $e = \{u, v\}$ of length l, with u the parent of v, we want, for $i = 1, \ldots, l-1$, to output as a new group those edges of length i that belong to the subtree of v, i.e., the maximal subtree of T_{M} rooted at v, and that were not output earlier. In addition, in order to output the last edges, we pretend that the search retreats from the root over an imaginary edge of length ∞. In order to implement the procedure, we could use an initially empty instance of the dictionary described in the beginning of the section and, in the situation above, push e on the stack of index l after popping each of the stacks of index $1, \ldots, l-1$ down to the level that it had when e was explored in the forward direction (in order to determine this, simply number the edges in the order in which they are encountered). Because of the effort involved in skipping stacks that, although nonempty, do not contain any edges sufficiently recent to be output, however, this would not work in linear time. In order to remedy this, we stay with a single array of stacks, but associate a bit vector with each node in T_{M}. When the search explores an edge $\{u, v\}$ of length l in the forward direction, with u the parent of v, the dictionary of v is initialized to all-zero (denoting an

empty set), and when the search retreats over $\{u, v\}$, the bit of index l is set in the bit vector of u, which is subsequently replaced by the bitwise OR of itself and the bit vector of v. Thus the final bit vector of v describes the part of the array of stacks more recent than the forward exploration of $\{u, v\}$, so that, when the search retreats over $\{u, v\}$, the relevant edge groups can be output in constant time plus time proportional to the size of the output. Overall, the depth-first search takes $O(n)$ time.

Define the length of a group output by the previous step as the common length of all edges in the group (their former level). We number the groups consecutively and create a node of level equal to the group length for each group and a node of level -1 for each vertex in V. Moreover, each group output when the search retreats over an edge e, except for the longest group output at the root, computes its parent group as the shortest group longer than itself among the group containing e and the groups output when the search retreats over e, and each vertex $v \in V$ computes its parent group as a shortest group containing an edge incident on v. This constructs a component tree for $\mathcal{N}$ in $O(n+m)$ time.

References

1. R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.
2. R. K. Ahuja, K. Mehlhorn, J. B. Orlin, and R. E. Tarjan, Faster algorithms for the shortest path problem, *J. ACM* **37** (1990), pp. 213–223.
3. R. Bellman, On a routing problem, *Quart. Appl. Math.* **16** (1958), pp. 87–90.
4. B. V. Cherkassky, A. V. Goldberg, and C. Silverstein, Buckets, heaps, lists, and monotone priority queues, *SIAM J. Comput.* **28** (1999), pp. 1326–1346.
5. G. B. Dantzig, On the shortest route through a network, *Management Sci.* **6** (1960), pp. 187–190.
6. E. W. Dijkstra, A note on two problems in connexion with graphs, *Numer. Math.* **1** (1959), pp. 269–271.
7. L. R. Ford, Jr. and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.
8. M. L. Fredman and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* **34** (1987), pp. 596–615.
9. M. L. Fredman and D. E. Willard, Trans-dichotomous algorithms for minimum spanning trees and shortest paths, *J. Comput. System Sci.* **48** (1994), pp. 533–551.
10. H. N. Gabow, A scaling algorithm for weighted matching on general graphs, in Proc. 26th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1985), pp. 90–100.
11. T. Hagerup, Sorting and searching on the word RAM, in Proc. 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS 1998), Lecture Notes in Computer Science, Vol. 1373, Springer, Berlin, pp. 366–398.
12. D. B. Johnson, Efficient algorithms for shortest paths in sparse networks, *J. ACM* **24** (1977), pp. 1–13.
13. R. C. Prim, Shortest connection networks and some generalizations, *Bell Syst. Tech. J.* **36** (1957), pp. 1389–1401.

14. R. Raman, Priority queues: Small, monotone and trans-dichotomous, in Proc. 4th Annual European Symposium on Algorithms (ESA 1996), Lecture Notes in Computer Science, Vol. 1136, Springer, Berlin, pp. 121–137.
15. R. Raman, Recent results on the single-source shortest paths problem, *SIGACT News* **28**:2 (1997), pp. 81–87.
16. R. Raman, Priority queue reductions for the shortest-path problem, in Proc. 10th Australasian Workshop on Combinatorial Algorithms (AWOCA 1999), Curtin University Press, pp. 44-53.
17. R. E. Tarjan, Efficiency of a good but not linear set union algorithm, *J. ACM* **22**, (1975), pp. 215–225.
18. M. Thorup, On RAM priority queues, in Proc. 7th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1996), pp. 59–67.
19. M. Thorup, Randomized sorting in $O(n \log \log n)$ time and linear space using addition, shift, and bit-wise boolean operations, in Proc. 8th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA 1997), pp. 352–359.
20. M. Thorup, Faster deterministic sorting and priority queues in linear space, Proc. 9th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 1998), pp. 550–555.
21. M. Thorup, Undirected single-source shortest paths with positive integer weights in linear time, *J. ACM* **46** (1999), pp. 362–394.
22. P. van Emde Boas, Preserving order in a forest in less than logarithmic time and linear space, *Inform. Process. Lett.* **6** (1977), pp. 80–82.
23. P. van Emde Boas, R. Kaas, and E. Zijlstra, Design and implementation of an efficient priority queue, *Math. Syst. Theory* **10** (1977), pp. 99–127.
24. P. D. Whiting and J. A. Hillier, A method for finding the shortest route through a road network, *Oper. Res. Quart.* **11** (1960), pp. 37–40.

Improved Algorithms for Finding Level Ancestors in Dynamic Trees

Stephen Alstrup and Jacob Holm

The IT University of Copenhagen,
Glentevej 67, DK-2400, Denmark.
{stephen,jholm}@itu.dk

Abstract. Given a node x at depth d in a rooted tree $LevelAncestor(x, i)$ returns the ancestor to x in depth $d - i$. We show how to maintain a tree under addition of new leaves so that updates and level ancestor queries are being performed in worst case constant time. Given a forest of trees with n nodes where edges can be added, m queries and updates take $O(m\alpha(m, n))$ time. This solves two open problems (P.F. Dietz, Finding level-ancestors in dynamic trees, LNCS, 519:32-40, 1991). In a tree with node weights, $min(x, y)$ report the node with minimum weight on the path between the nodes x and y. We can substitute the $LevelAncestor$ query with min, without increasing the complexity for updates and queries. Previously such results have been known only for special cases (e.g. R.E. Tarjan. Applications of path compression on balanced trees. J.ACM, 26(4):690-715, 1979).

1 Introduction

Given a collection of rooted trees and a node x in depth d, $LevelAncestor(x, i)$ returns the ancestor to x in depth $d-i$. We give a simple algorithm to preprocess a tree in linear time so that queries can be answered in worst case constant time. New leaves can be added to the tree ($AddLeaf$), so that each update and query take worst case constant time. For a forest of trees with n nodes where new edges may be added ($Link$), updates have amortized complexity $O(a(l, n))$ and queries have worst case complexity $O(l)$ time, where a is the row inverse Ackerman function and $l > 0$ is an arbitrary integer. This matchs a RAM lower bound [1], for word size $\Theta(\log n)$. The results are presented in a self-contained manner, i.e., the results use classic techniques, but do not depend on any non-trivial data structures.

In [9], Chazelle needs a specialized version of the problem: given two nodes x and y, return the child of y which is an ancestor to x. An identical problem arises in range query problems [17]. In [19], Harel and Tarjan give a quite involved optimal algorithm for the level ancestor problems for a special kind of trees with maximum height $O(\log n)$. The solution is used to find nearest common ancestors of two nodes in a tree. In [21], level ancestors queries are used to recognize breadth-first trees, and Dietz [10] studies the problem in connection with persistent data structures.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 73–84, 2000.
© Springer-Verlag Berlin Heidelberg 2000

In [7], Berkman and Vishkin show how to preprocess a tree in linear time so that level ancestors can be answered on-line in constant time, however, introducing a constant $2^{2^{28}}$. The solution in [7] is based on a reduction using an Euler tour in the tree. Dietz [10] gave an amortized constant time per operation algorithm for a tree that grows under addition of leaves. Doing this he uses an approach different from [7]. Dietz partition the tree into heavy paths and uses a fast search structure for small sets [13] in a two level system. The fast search structure supports predecessor queries in constant time in a set with at most $O(\log n)$ keys.

Our worst case optimal algorithm for adding leaves to a tree, and our optimal result for maintaining a forest where new edges can be inserted, solve two open problems stated by Dietz in [10]. The results that we present apply another approach than Berkman, Vishkin and Dietz [7,10], and does not use a fast search structure or introduce large constants. For a static tree and a tree which can grow under the addition of leaves, we present a new shortcutting technique and combine it with a version of Gabow and Tarjan's [16] micro tree technique similar to that it [4]. To maintain a forest under insertion of new edges, we use Gabow's α-scheme [14,15]. Doing this in a direct way introduce a more general level ancestor problem where edges have associated weights. We show how to solve this problem, still keeping the constants low.

1.1 Variants and Extensions

Let the nodes in the tree have weights associated. We can substitute the *LevelAncestor* query with the queries below, without increasing the complexity for updates and queries:

- $min(x, y)$: return the node with minimum weight on the path $x \cdots y$.
- $succ(x, y, d)$: return the first node z on the path from x to y where $dist(x, z) \geq d$. Here $dist$ is the sum of weights on the path between the two nodes.

For the *succ* operation the weights must be polylogarithmically bounded integers, i.e., $weight(v) = O((\log n)^c)$ for a constant c, to achieve the claimed bounds. For larger weights we are dealing with the classic predecessor problem [5]. To achieve the claimed bounds for *min*, we need the ability to determine the rank of a weight in a small dynamic set. To do this we use the results from [13].

In [9,26], Chazelle and Thorup (in a parallel computation model) show how to insert m shortcut edges in a tree so that given any pair of nodes, a path (using shortcut edges) of length at most $O(\alpha(m, n))$ can be reported in a time linear in the length of the path. Their results are optimal by a result of Yao [30]. For each shortcut edge (a, b) inserted, two (shortcut) edges $(a, c), (c, d)$ already exist. Their technique only applies to static trees, i.e., addition of new leafs is not allowed. Using the shortcutting technique, *min* queries can be answered in $O(\alpha(n, n))$ time after linear time preprocessing. The problem of determining

wheter a spanning tree is a minimum spanning tree for a graph, can be reduced to the *min* query for a static tree. Using the shortcutting technique, an almost linear time algorithm is achieved. As a direct application of the classic Union-Find technique for disjoint sets, Tarjan [24] shows how to answer queries like *min* for a forest, which can be updated under edge insertions. However, queries are restricted to include the root of the tree, i.e. finding the minimum weight from a node to the root. Linear time minimum spanning tree verification algorithms [8,11,20] take advantage of the fact that all queries are known in advance. This, combined with micro tree techniques [16] and [24], gives a linear time algorithm. In [20] it was raised as an open question how to preprocess a tree in linear time so that *min* queries could be answered in constant time. Finding the minimum weight on just a path for a restricted domain $(1 \cdots n)$ is sufficient to find nearest common ancestor (NCA) in a tree (see [17]). Two optimal (almost) identical algorithms for NCA in a static tree are given in [19,22], whereas [6] solves the *min* path problem. In [18], Harel considers *min* queries in a forest where new edges can be inserted as in [24]. He gives a linear time algorithm for the case where all edge insertions are known in advance and the weights can be sorted in linear time.

Summarizing : The static techniques for shortcutting are more general than our techniques, but use optimal $O(\alpha(n, n))$ time for each query compared to our constant time complexity. Our techniques can handle addition of leaves in worst case constant time. For the more general *link* command we support queries between any pair of nodes in the tree, opposite to Tarjan, where one of the nodes should be the root, thus somehow restricting the technique to off-line problems. Furthermore, our techniques support the *succ* command as opposed to both Tarjan's and the static shortcutting techniques. The *succ* command is used for *LevelAncestor*-like problems, and in the more general cases also for successor/predecessor queries in a tree. In order to answer *min* queries in constant time after linear time preprocessing we need non-comparison based techniques [13].

1.2 Fully Dynamic Trees

In [23], Sleator and Tarjan give an algorithm to maintain a forest of trees under insertion and deletion of new edges supporting *min* queries. Each operation is supported in worst case $O(\log n)$ per operation. As trivial applications of top trees [2,3] *LevelAncestor* and *succ* can also be supported in $O(\log n)$ worst case time per operation. On a RAM with word size $\Theta(\log n)$, we have the usual gap to the $\Omega(\log n / \log\log n)$ lower bound [12] for fully dynamic tree problems. For the results achieved in [23] and the applications of top trees [2] there is no restriction on the node weights. Both algorithms are pointer algorithms [25] and optimal for this model, since we for the static case have a trivial lower bound $\Omega(\log h)$ for queries in a tree with height h. A matching pointer algorithm upper bound for the static problem is given by Tsakalidis and Van Leeuwen [27,28,29].

1.3 Outline

In section 2 we give a linear time algorithm to preprocess a tree so that level ancestor queries can be answered in worst case constant time. Using the techniques from section 2, we in section 3 show how to support $AddLeaf$. Given this result in details, we sketch (because of lack of space) the remaining results in section 4.

1.4 Preliminaries

Let T be a rooted tree with root node $r = root(T)$. The *parent* of a node v in T is denoted by $parent(v)$. The depth of v is $d(v) = d(parent(v)) + 1$, and $d(r) = 0$. The nodes on the path from v to r are *ancestors* of v, and the nodes on the path from $parent(v)$ to r are *proper ancestors* of v. If w is an ancestor of v, then v is a *descendant* of w. $LevelAncestor(x, i)$ returns the ancestor y to x where $d(x) - d(y) = i$. If such an ancestor does not exists the root r is returned (in this paper we ignore this trivial case). The *subtree* rooted at v is the tree induced by all descendants of v. The size of the subtree rooted at v is denoted $s(v)$. The length of a path between two nodes v, w is denoted $dist(v, w)$ and is the number of nodes on the unique simple path between v and w. If the nodes/edges have weights, we let $dist(v, w)$ denote the sum of weights on the path. The notation $d|a$ means that $a = kd$ for an integer k. If such a k does not exist, we write $d \nmid a$. We let $\log x$ denote $\log_2 x$. For the row inverse Ackerman function a and the (functional) inverse Ackerman function α, we use the standard definition [15], i.e., $a(1, n) = O(\log n)$, $a(2, n) = O(\log^* n)$, etc.

2 Static Level-Ancestor

In this section we will show how to preprocess a tree in linear time and space, so that level ancestor queries can be answered on-line in worst case constant time. We do this by first introducing a simple algorithm which uses $O(n \log n)$ time and space for preprocessing of a tree with n nodes, and then using micro trees to reduce the time and space to linear.

2.1 Macro Algorithm

We define the *rank* of v, denoted $r(v)$, to be the maximum integer i so that $2^i \mid d(v)$ and $s(v) \geq 2^i$. Note that with this definition the rank of the root is $\lfloor \log_2 n \rfloor$.

Observation 1. *The number of nodes with rank $\geq i$ is at most $\lfloor n/2^i \rfloor$.*

The preprocessing algorithm consists of precomputing the depth, size and rank of each node in the tree and then constructing the following two tables:

`levelanc`$[v][x]$**:** contains the x'th ancestor to v, for $0 \leq x < 2^{r(v)}$.

`jump`$[v][i]$**:** contains the first proper ancestor to v whose depth is divisible by 2^i, for $0 \leq i < \log_2(d(v)+1)$.

The idea behind the algorithm is to use the `jump`[][] table a constant number of times in order to reach a node w with a sufficiently large rank to hold the answer to the level ancestor query in its table. The complete pseudocode for answering level ancestor queries looks as follows:

LevelAncestor(v, x)**:**
  $i := \lfloor \log_2(x+1) \rfloor$
  $d := d(v) - x$
  while $2(d(v) - d) \geq 2^i$ // max 4 times!
    $v :=$ `jump`$[v][i-1]$
  return `levelanc`$[v][d(v) - d]$

Lemma 2. *A tree with n nodes can be preprocessed in $O(n \log n)$ time and space, allowing level ancestor queries to be answered in worst case constant time.*

Proof. The depth and size of each node can be computed by a simple top-down/bottom-up traversal of the tree. For any node v, $r(v)$ can then be computed as $\max\{i : 2^i \mid d(v) \wedge s(v) \geq 2^i\}$, in $O(\log n)$ time.

The `levelanc`$[v]$[] table can for each v be constructed in $O(2^{r(v)})$ time, by following parent pointers. The total time to build the `levelanc`[][] table is therefore linear in the number of entries. By observation 1, the number of entries is less than $\sum_{0 \leq i < \log_2 n} \lfloor \frac{n}{2^i} \rfloor 2^i \leq n \log_2 n$.

For any v and any i, $0 \leq i < \log_2(d(v)+1)$, we note that `jump`$[v][i] =$ parent(v) if $2^i \mid d(\text{parent}(v))$, and `jump`$[v][i] =$ `jump`$[\text{parent}(v)][i]$ otherwise. Thus the `jump`[][] table can be computed by a simple top-down traversal of the tree, and like above this takes linear time in the number of entries, which is $O(n \log n)$.

Now we only need to show that a LevelAncestor(v, x) query is computed in worst case constant time. If $x = 0$ or $x = 1$ this is trivial, so assume that $x > 1$. Let w be the ancestor of v with depth $d = d(v) - x$. Then w is the node we should return. Setting $i := \lfloor \log_2(x+1) \rfloor$ as in the algorithm, we have that among the $x+1$ nodes on the path from v to w, there are between 2 and 4 nodes whose depth is divisible by 2^{i-1}. Obviously the "while" loop finds the topmost of these in at most four steps, and then it stops. After the loop, $d(v)$ is divisible by 2^{i-1}, and since we know that v has descendants down to $d(v) + 2^{i-1}$, we must also have $s(v) \geq 2^{i-1}$. Thus $r(v) \geq i - 1$ and the `levelanc`$[v]$[] table has length $\geq 2^{i-1}$. But since $d(v) - d < 2^{i-1}$, we can now find w as `levelanc`$[v][d(v) - d]$. Thus, as desired, we have found w in at most a constant number of steps.

2.2 Hybrid Algorithm

In this section we will reduce the time and space complexity of our preprocessing algorithm to linear by applying the following lemma, proved in the next section.

Lemma 3. *Given $O(n)$ time and space for pre-preprocessing, we can preprocess any tree T with at most $\frac{1}{2}\log_2 n$ nodes in $O(|T|)$ time, allowing level ancestor queries to be answered in worst case constant time.*

Let T be a tree with $n \geq 4$ nodes, let $r_0 = \lfloor \log_2 \log_2 n - 1 \rfloor$ and let $M = 2^{r_0}$. Now r_0 and M are integers and $\frac{1}{4}\log_2 n < M \leq \frac{1}{2}\log_2 n$. We define the set of *macro* nodes in T to be the set of all nodes v in T with $r(v) \geq r_0$. By observation 1 there are at most $n/2^{r_0} = n/M$ macro nodes in T.

Observation 4. *Let v be a node with $s(v) \geq M$, then there is a macro node among the first M ancestors to v.*

To enable finding the first macro node that is a proper ancestor to any node, we introduce the following table:

`jumpM`$[v]$**:** contains the first proper ancestor of v whose depth is divisible by M.

The first proper macro node ancestor to a node v is then either `jumpM`$[v]$ or `jumpM`[`jumpM`$[v]$]. (The reason for not simply letting `jumpM`$[v]$ point directly to the node is to simplify section 3.2).

Let T/M denote the *macro* tree induced by the macro nodes of T. Since $M = \Omega(\log n)$, we can use the algorithm from the previous section to preprocess T/M in $O(n/M \log(n/M)) = O(n)$ time and space to answer level ancestor queries in constant time. The distance in T between any macro node and its parent in the macro tree T/M is exactly M; thus for any macro node v we can in constant time find the macro node of least depth on the path in T from v to $\text{LevelAncestor}(v, x)$ by simply computing $\text{LevelAncestor}_{T/M}(v, \lfloor \frac{x}{M} \rfloor)$. (Even the division takes constant time, since $M = 2^{r_0}$ is a power of 2). The distance from this node to $\text{LevelAncestor}(v, x)$ is less than M.

If there is a macro node on the path from v to $w = \text{LevelAncestor}(v, x)$, we therefore only need to find the first macro node that is ancestor to v and use the algorithm for the macro tree to find a node with no macro nodes on the path to w. The above discussion shows that this can be done in worst case constant time. Thus all that remains to be shown is how to handle level ancestor queries where there are no macro nodes on the path. The idea now is to partition T into *micro* trees, in scuh a way that we can find any of the ancestors to a node v up to the first macro node by looking in at most two micro trees. Specifically we partition T into micro trees of size $\leq M$, so that if $|\mu| < M$ for some micro tree μ, then all descendants of $\text{root}(\mu)$ are in μ. The partition can be done in linear time using one top-down traversal of the tree. A micro tree in this partition is called full, if it has exactly M nodes. For any node v, let $\mu(v)$ denote the micro tree containing v, and let $\mu_p(v)$ denote the micro tree $\mu(\text{parent}(\text{root}(\mu(v))))$. From the definition of the partition it follows that $\mu_p(v)$ is a full tree, unless $\text{root}(T) \in \mu(v)$ (in which case $\mu_p(v)$ is undefined).

For each micro tree μ create a table `levelancM`$[\mu][]$ containing the first $|\mu|$ ancestors to $\text{root}(\mu)$. Since the micro trees form a partition of T, this table has exactly one entry for each node in T and thus size $O(n)$. By observation 4 the

`levelancM`[][] table for a full micro tree contains a macro node. It follows that each ancestor of a node v up to the first macro node is contained in either $\mu(v)$, $\mu_p(v)$ or `levelancM`$[\mu_p(v)]$[] as desired. By lemma 3, level ancestor queries can now be answered in worst case constant time as follows.

LevelAncestor(v, x)**:**
  $d := d(v) - x$
  $w :=$ `jumpM`$[v]$
  if w is not a macro node
    $w :=$ `jumpM`$[w]$
  // Now w is the first macro node on the path from v to root(v).
  if $d(w) > d$ then
    $v :=$ LevelAncestor$_{T/M}(w, \lfloor \frac{d(w)-d}{M} \rfloor)$
  // Now there are no macro nodes on the path from v.
  if $d(\text{root}(\mu(v))) \leq d$ then
    return LevelAncestor$_{\mu(v)}(v, d(v) - d)$
  $v :=$ parent$(\text{root}(\mu(v)))$
  // $\mu(v)$ is now a full micro tree.
  if $d(\text{root}(\mu(v))) \leq d$ then
    return LevelAncestor$_{\mu(v)}(v, d(v) - d)$
  return `levelancM`$[\mu(v)][d(\text{root}(\mu(v))) - d]$

We have thus proven the following:

Theorem 5. *A tree can be preprocessed in linear time and space, allowing level ancestor queries to be answered in worst case constant time.*

2.3 Micro Algorithm

In this section we will show how to construct a set of tables in $O(n)$ time, so that we can preprocess any tree with at most $N = \lfloor \frac{1}{2} \log_2 n \rfloor$ nodes, allowing level ancestor queries to be answered in worst case constant time.

Let μ be the micro tree we want to preprocess. We number all the nodes of μ in a top-down order and create a table `nodetable`$[\mu][i]$ containing the nodes in μ with number i for each $0 \leq i < |\mu|$. To represent the ancestor relation efficiently, we use a single word `anc`$[v]$ for each node v, where bit i is set if and only if the node numbered i in μ is an ancestor of v. To find the xth ancestor to v we now only need to find the index i of the xth most significant bit that is set in `anc`$[v]$ and then return `nodetable`$[\mu][i]$. In order for this query to take worst case constant time, we construct the following table which is completely independent of μ:

`bitindex`$[w][i]$**:** contains the position of the ith most significant set bit in w, for $0 \leq w < 2^N$ and $0 \leq i < N$. If only $k < i + 1$ bits of w are set, `bitindex`$[w][i] = k - (i + 1)$.

Given these tables, each level ancestor query is answered in worst case constant time as follows:

LevelAncestor$_\mu(v, x)$**:**
$i :=$ `bitindex`[`anc`$[v]$][x]
if $i > -1$ return `nodetable`$[\mu(v)][i]$

We are now ready to prove lemma 3:

Lemma. *Given $O(n)$ time and space for pre-preprocessing, we can preprocess any tree T with at most $\frac{1}{2}\log_2 n$ nodes in $O(|T|)$ time, allowing level ancestor queries to be answered in worst case constant time.*

Proof. First, we must show how to create the `bitindex`[][] table. This can easily be done, as `bitindex`$[2^j+k][0] = k$, and `bitindex`$[2^j+k][i] =$ `bitindex`$[k][i-1]$ for $0 \leq k < 2^j < 2^N$ and $0 < i < N$. This means that we can start by setting `bitindex`$[0][i] := -(i+1)$ for $0 \leq i < N$, and then keep doubling the table size until we are done. This takes linear time in the number of entries, which is $N2^N \leq \frac{1}{2}\log_2 n 2^{\frac{1}{2}\log_2 n} = \frac{1}{2}n^{\frac{1}{2}}\log_2 n = O(n)$. This concludes the pre-preprocessing.

To preprocess each micro tree μ with at most N nodes, we do as follows: Traverse μ in any top-down order. Let i be the number of the node v in the traversal. We set `nodetable`$[\mu][i] := v$ and if v is the root of μ we set `anc`$[v] := 2^i$; otherwise we set `anc`$[v] :=$ `anc`$[\text{parent}(v)] + 2^i$.

3 Level Ancestor with AddLeaf

In this section we will extend the static level ancestor algorithm from the previous section to support the *AddLeaf* operation in worst case constant time. We do this in three steps: First we show that the macro algorithm can be extended to support *AddLeaf* operations in worst case logarithmic time. Second, we show that the selection of macro nodes in the hybrid algorithm can be done in worst case constant time per *AddLeaf*. Finally we show that the micro algorithm can be modified to allow *AddLeaf* operations in worst case constant time.

For both the micro- and macro algorithms we implicitly use the well-known result that we can maintain a dynamically sized array supporting each of the operations IncreaseLength (adding one to the length of the array) and LookUp (returning the i'th element in the array) in worst case constant time.

3.1 Macro Algorithm

When a new leaf v is added to the tree T, we must make sure that the tables `levelanc`[][] and `jump`[][] are updated correctly. For the node v `jump`$[v]$[] can be computed exactly as in the static case in $O(\log n)$ time and `levelanc`$[v]$[] have only one entry `levelanc`$[v][0] = v$. The addition of v does not influence the `jump`[][] tables for any other node in the tree, but it may cause the rank of $\lfloor \log_2 n \rfloor$ of its ancestors to increase by one, which means that the `levelanc`[][] tables for these ancestors should be doubled.

If we just doubled the `levelanc`[][] tables as necessary, we would get an algorithm with amortized complexity $O(\log n)$ per *AddLeaf*. Instead, when we add a new leaf v to the structure, we extend the `levelanc`$[w]$[] table for each ancestor w to whose rank v may contribute. The rank of w can only be increased when the number of nodes below w has doubled since the last increase; thus when the rank *is* increased, the table already has the correct length. The new node v has at most $\lfloor \log_2(d(v)+1) \rfloor$ ancestors whose rank it may contribute to, and these are exactly the nodes in the `jump`$[v]$[] table. Thus, when we add v as a new leaf, only a logarithmic number of tables has to be extended. Using the standard table-extending technique mentioned earlier, this can be done in worst case logarithmic time.

The following macro algorithm for *AddLeaf* therefore runs in worst case logarithmic time:

AddLeaf(v, p)**:**
  `levelanc`$[v][0] := v$
  for $i := 0$ to $\log_2(d(v)+1) - 1$ do
    if $2^i \mid d(p)$ then $w := p$ else $w :=$ `jump`$[p][i]$
    `jump`$[v][i] := w$
    extend the `levelanc`$[w]$[] table with one more ancestor to w.

As a final remark, we note that this algorithm can be run incrementally, dividing each *AddLeaf* into $O(\log n)$ separate steps (one for each iteration of the for-loop). The AddLeafSteps for different nodes can be mixed arbitrarily (however new leaves can only be added below fully inserted nodes) together with LevelAncestor queries concerning fully inserted nodes. Each AddLeafStep and LevelAncestor query still runs in worst case constant time, and this will be important in the next section.

3.2 Hybrid Algorithm

In order to extend the hybrid algorithm from section 2.2 to allow *AddLeaf* operations, we must show how to maintain both the `jumpM`[] and `levelancM`[][] tables. Adding a leaf to the tree does not change `jumpM`$[w]$ for any node w, so we only have to compute `jumpM`$[v]$ for the new leaf v, and this takes constant time.

To maintain the `levelancM`[][] table when adding a leaf v with parent p in the tree, we have two cases.

1. If $|\mu(p)| < M$ we must add v as a leaf in $\mu(p)$ and extend the `levelancM`$[\mu(p)]$[] table.
2. Otherwise create a new micro tree $\mu(v)$ and set `levelancM`$[\mu(v)][0] := v$.

The rest of the work is done either by the micro algorithm $AddLeaf_\mu$ described in the next section or by the AddLeafStep algorithm from the previous section. If each runs in worst case constant time, we can combine them into the following hybrid algorithm running in worst case constant time:

AddLeaf(v, p)**:**
  AddLeaf$_\mu(v, p)$
  if $2^{r_0} \mid d(p)$ then `jumpM`$[v] := p$ else `jumpM`$[v] :=$ `jumpM`$[p]$
  Add-LeafStep(`jumpM`$[v]$, `jumpM`[`jumpM`$[v]$])

Here it is assumed that it takes at most M AddLeafSteps to fully insert a node in the macro tree. The algorithm assumes that we know the total number of nodes and thus r_0 and M ahead of time. However, using standard doubling techniques, we can easily extend the algorithm to handle the case where the total number of nodes is not known in advance.

Theorem 6. *We can maintain a rooted tree supporting AddLeaf and LevelAncestor operations in worst case constant time.*

3.3 Micro Algorithm

When adding a leaf to a micro tree we only need to show how to update the `nodetable`[][] and `anc`[] tables, since the `bitindex`[][] table depends only on the size of M, and can be handled using standard doubling techniques. The full micro algorithm for *AddLeaf* looks as follows:

AddLeaf$_\mu(v, p)$**:**
  $k := |\mu|$
  extend `nodetable`$[\mu]$[] with v.
  `anc`$[v] :=$ `anc`$[p] + 2^k$

4 Link and Querie Variants

In this section we sketch how to support insertion of new edges in a forest of rooted trees supporting level ancestor, *min* and *succ* queries. First we focus on level ancestor. Let r be the root of a tree T and v a new node. The operation *AddRoot*(v, r) inserts a new edge between the node v and r, making v the root of the combined tree. Hence, the depth of all nodes in T increases by 1, and $d(r) = 1$ after the operation. Let A be the algorithm given in the last section supporting addition of new leaves in a tree in constant time. It is simple to extend A, so that the operation *AddRoot*(v, r) is supported in worst case constant time, using an extendable array for all nodes added as roots to the tree. In general we have:

Theorem 7. *We can maintain a dynamic forest of rooted trees supporting AddLeaf, AddRoot and LevelAncestor operations in worst case constant time.*

In [14,15] Gabow gives an α-scheme to handle insertion of new edges in a forest of rooted trees for connectivity-like queries. In order to use the α-scheme one should provide an algorithm which handles *AddLeaf* and *AddRoot* operations. Given such an algorithm with constant time worst case complexity for

updates and queries, the α-scheme (if it can be applied) gives an algorithm with amortised complexity $O(a(l,n))$ for updates and worst case complexity $O(l)$ for queries. The approach is to essentially insert shortcut edges (corresponding to path compression in Union-Find algorithms) in the tree. For connectivity this is straightforward, since the goal is to report the root of the tree. For the level ancestor problem, the shortcut edges introduce edges with weights in the tree. The edge weights change the problem to the problem of answering *succ* queries. However, by applying Gabow's technique carefully, it is possible to limit the edge weights to $O((\log n)^2)$. In general we can handle edge weights of size $O(polylogn)$ using micro tree techniques. The basic idea is to use one level of micro trees to reduce the edge weights from $O(\log_2^k n)$ to $O(\log_2^{k-1} n)$. Doing this k times only increases space and time by a factor k.

Theorem 8. *For any $l > 0$, we can maintain a forest with n nodes, supporting Link in $O(a(l,n))$ amortized time and LevelAncestor in $O(l)$ worst case time.*

Using the edge weights reduction technique, the *succ* operation can be supported in a static tree and in dynamic forest in the same time as *LevelAncestor*, if the edge weights are polylogarithmically bounded positive integers.

In order to answer *min* queries we use the following observation from the level ancestor algorithm: Essentially the level ancestor algorithm consists of shortcutting edges (from a jump table) and special treatment of micro trees. When constructing a shortcutting edge we can in the same time associate the minimum weight the shortcut edge covers reducing the problem to micro trees. In order to use the micro tree techniques presented in this paper for *min* queries, we need to know the rank for each weight in a micro tree. Since a micro tree has at most $O(\log n)$ edges, we can use fast search structures [13] to find the rank of any weight in constant time. Thus, we conclude, the *min* operation can be supported in a static tree and dynamic forest in the same time as *LevelAncestor*, if non-comparison techniques are allowed.

References

1. S. Alstrup, A. Ben-Amram, and T. Rauhe. Worst-case and amortised optimality in union-find. In *31th ACM Symp. on Theo. of Comp.*, pages 499–506, 1999.
2. S. Alstrup, J. Holm, K. de Lichtenberg, and M. Thorup. Minimizing diameters of dynamic trees. In *24th Int. Col. on Auto., Lang. and Prog.*, volume 1256 of *LNCS*, pages 270–280, 1997.
3. S. Alstrup, J. Holm, and M. Thorup. Maintaining center and median in dynamic trees. In *7th Scan. Work. on Algo. Theo.*, LNCS, 2000.
4. S. Alstrup and M. Thorup. Optimal pointer algorithms for finding nearest common ancestors in dynamic trees. In *5th Scan. Work. on Algo. Theo.*, volume 1097 of *LNCS*, pages 212–222, 1996. To appear in J. of Algo.
5. P. Beame and F. Fich. Optimal bounds for the predecessor problem. In *31th ACM Symp. on Theo. of Comp.*, pages 295–304, 1999.
6. O. Berkman and U. Vishkin. Recursive star-tree parallel data structure. *SIAM J. on Comp.*, 22(2):221–242, 1993.

7. O. Berkman and U. Vishkin. Finding level-ancestors in trees. *J. of Comp. and Sys. Scie.*, 48(2):214–230, 1994.
8. A. Buchsbaum, H. Kaplan, A. Rogers, and J. Westbrook. Linear-time pointer-machine algorithms for lca's, mst verification, and dominators. In *30th ACM Symp. on Theo. of Comp.*, pages 279–288, 1998.
9. B. Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2:337–361, 1987.
10. P. Dietz. Finding level-ancestors in dynamic trees. In *2nd Work. on Algo. and Data Struc.*, volume 1097 of *LNCS*, pages 32–40, 1991.
11. B. Dixon, M. Rauch, and R. E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM J. on Comp.*, 21(6):1184–1192, 1992.
12. M. Fredman and M. Saks. The cell probe complexity of dynamic data structures. In *21st ACM Symp. on Theory of Comp.*, pages 345–354, 1989.
13. M. Fredman and D. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. of Comp. and Sys. Sci.*, 48(3):533–551, 1994.
14. H. Gabow. A scaling algorithm for weighted matching on general graphs. In *26th Symp. on Found. of Comp. Sci.*, pages 90–100, 1985.
15. H. Gabow. Data structure for weighted matching and nearest common ancestors with linking. In *1st Symp. on Dis. Algo.*, pages 434–443, 1990.
16. H. Gabow and R. Tarjan. A linear-time algorithm for a special case of disjoint set union. *J. of Comp. and Sys. Sci.*, 30(2):209–221, 1985.
17. H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for computational geometry. In *16th ACM Symp. on Theo. of Comp.*, pages 135–143, 1984.
18. D. Harel. A linear time algorithm for finding dominator in flow graphs and related problems. In *17th ACM Symp. on Theo. of Comp.*, pages 185–194, 1985.
19. D. Harel and R. Tarjan. Fast algorithms for finding nearest common ancestors. *Siam J. on Comp.*, 13(2):338–355, 1984.
20. V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18(2):263–270, 1997.
21. U. Manber. Recognizing breadth-first search trees in linear time. *Information Processing Letters*, 34(4):167–171, 1990.
22. B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. on Comp.*, 17:1253–1262, 1988.
23. D. Sleator and R. Tarjan. A data structure for dynamic trees. *J. of Comp. and Sys. Sci.*, 26(3):362–391, 1983. See also STOC 1981.
24. R. Tarjan. Applications of path compression on balanced trees. *J. of the ACM*, 26(4):690–715, 1979.
25. R. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *J. of Comp. and Sys. Sci.*, 18(2):110–127, 1979. See also STOC 1977.
26. M. Thorup. Parallel shortcutting of rooted trees. *J. of Algo.*, 23(1):139–159, 1997.
27. A. Tsakalidis. The nearest common ancestor in a dynamic tree. *Acta informatica*, 25(1):37–54, 1988.
28. A. Tsakalidis and J. van Leeuwen. An optimal pointer machine algorithm for finding nearest common ansectors. Technical Report RUU-CS-88-17, Dep. of Comp. Sci., Uni. of Utrecht, 1988.
29. J. van Leeuwen. Finding lowest common ancestors in less than logarithmic time. Unpublish technical report, 1976.
30. A. Yao. Space-time tradeoff for answering range queries. In *14th ACM Symp. on Theo. of Comp.*, pages 128–136, 1982.

Lax Logical Relations

Gordon Plotkin[1,⋆], John Power[1,⋆⋆],
Donald Sannella[1,⋆⋆⋆], and Robert Tennent[1,2,†]

[1] Laboratory for Foundations of Computer Science
University of Edinburgh, Edinburgh, U.K. EH9 3JZ
{gdp,ajp,dts}@dcs.ed.ac.uk
[2] Department of Computing and Information Science
Queen's University, Kingston, Canada K7L 3N6
rdt@cs.queensu.ca

Abstract. Lax logical relations are a categorical generalisation of logical relations; though they preserve product types, they need not preserve exponential types. But, like logical relations, they are preserved by the meanings of all lambda-calculus terms. We show that lax logical relations coincide with the correspondences of Schoett, the algebraic relations of Mitchell and the pre-logical relations of Honsell and Sannella on Henkin models, but also generalise naturally to models in cartesian closed categories and to richer languages.

1 Introduction

Logical relations and various generalisations are used extensively in the study of typed lambda calculi, and have many applications, including

- characterising lambda definability [Pl73, Pl80, JT93, Al95];
- relating denotational semantic definitions [Re74, MS76];
- characterising parametric polymorphism [Re83];
- modelling abstract interpretation [Ab90];
- verifying data representations [Mi91];
- defining fully abstract semantics [OR95]; and
- modelling local state in higher-order languages [OT95, St96].

The two key properties of logical relations are

1. the so-called Basic Lemma: a logical relation is preserved by the meaning of every lambda term; and
2. inductive definition: the type-indexed family of relations is determined by the base-type components.

⋆ This author was supported by EPSRC grants GR/J84205 and GR/M56333.
⋆⋆ This author was supported by EPSRC grants GR/J84205 and GR/M56333, and by a grant from the British Council.
⋆⋆⋆ This author was supported by EPSRC grant GR/K63795.
† This author was supported by a grant from the Natural Sciences and Engineering Research Council of Canada.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 85–102, 2000.
© Springer-Verlag Berlin Heidelberg 2000

It has long been known that there are type-indexed families of conventional relations that satisfy the Basic Lemma but are *not* determined inductively in a straightforward way. Schoett [Sc87] uses families of relations that are preserved by algebraic operations to treat behavioural inclusion and equivalence of algebraic data types; he terms them "correspondences," but they have also been called "simulations" [Mi71] and "weak homomorphisms" [Gi68]. Furthermore, Schoett conjectures (pages 280–81) that the Basic Lemma will hold when appropriate correspondences are used between models of lambda calculi, and that such relations compose. Mitchell [Mi90, Sect. 3.6.2] terms them "algebraic relations," attributing the suggestion to Gordon Plotkin[1] and Samson Abramsky, independently, and asserts that the Basic Lemma is easily proved and (binary) algebraic relations compose. But Mitchell concludes that, because logical relations are easily constructed by induction on types, they "seem to be the important special case for proving properties of typed lambda calculi."

Recently, Honsell and Sannella [HS99] have shown that such relation families, which they term "pre-logical relations," are both the *largest* class of conventional relations on Henkin models that satisfy the Basic Lemma, and the smallest class that both includes logical relations and is closed under composition. They give a number of examples and applications, and study their closure properties.

We briefly sketch two of their applications.

- The composite of (binary) logical relations need not be logical. It is an easy exercise to construct a counter-example; see, for instance, [HS99]. But the composite of binary pre-logical relations *is* a pre-logical relation.
- Mitchell [Mi91] showed that the use of logical relations to verify data representations in typed lambda calculi is complete, provided that all of the primitive functions are first-order. In [HS99], this is strengthened to allow for higher-order primitives by generalising to *pre*-logical relations. Honsell, Longley et al. [HL$^+$] give an example in which a pre-logical relation is used to justify the correctness of a data representation that cannot be justified using a conventional logical relation.

In this work, we give a *categorical* characterisation of algebraic relations (simulations, correspondences) between Henkin models of typed lambda calculi. The key advantage of this characterisation is its generality. By using it, one can immediately generalise from Henkin models to models in categories very different from *Set*, and to languages very different from the simply typed lambda calculus, for example to languages with co-products or tensor products, or to imperative languages without higher-order constructs.

The paper is organised as follows. In Sect. 2, we recall the definition of logical relation and a category theoretic formulation. In Sect. 3, we give our categorical notion of lax logical relation, proving a Basic Lemma, with a converse. In Sect. 4, we explain the relationship with pre-logical relations and in Sect. 5 give another syntax-based characterisation. In Sect. 6 we consider models in cartesian closed categories. In Sect. 7, we generalise our analysis to richer languages.

[1] Plotkin recalls that the suggestion was made to him by Eugenio Moggi in a conversation.

2 Logical Relations

Let Σ be a signature of basic types and constants for the simply typed λ-calculus with products [Mi90], generating a language L. We use σ and τ to range over types in L. We denote the set of functions from a set X to a set Y by $[X \Rightarrow Y]$.

Definition 2.1. *A* model M *of* L *in Set consists of*

- *for each* σ*, a set* M_σ*, such that* $M_{\sigma\to\tau} = [M_\sigma \Rightarrow M_\tau]$*,* $M_{\sigma\times\tau} = M_\sigma \times M_\tau$*, and* $M_1 = \{*\}$*;*
- *for each constant* c *of* Σ *of type* σ*, an element* $M(c)$ *of* M_σ*.*

A model extends inductively to send every judgement $\Gamma \vdash t\colon \sigma$ of L to a function $M(\Gamma \vdash t\colon \sigma)$ from M_Γ to M_σ, where M_Γ is the evident finite product in *Set*. These are "full" type hierarchies; larger classes of models, such as Henkin models and cartesian closed categories, will be discussed later.

Definition 2.2. *Given a signature* Σ *and two models,* M *and* N*, of the language* L *generated by* Σ*, a* (binary) logical relation from M to N *consists of, for each type* σ *of* L*, a relation* $R_\sigma \subseteq M_\sigma \times N_\sigma$ *such that*

- *for all* $f \in M_{\sigma\to\tau}$ *and* $g \in N_{\sigma\to\tau}$*, we have* $f\, R_{\sigma\to\tau}\, g$ *if and only if for all* $x \in M_\sigma$ *and* $y \in N_\sigma$*, if* $x\, R_\sigma\, y$ *then* $f(x)\, R_\tau\, g(y)$*;*
- *for all* $(x_0, x_1) \in M_{\sigma\times\tau}$ *and* $(y_0, y_1) \in N_{\sigma\times\tau}$*, we have* $(x_0, x_1)\, R_{\sigma\times\tau}\, (y_0, y_1)$ *if and only if* $x_0\, R_\sigma\, y_0$ *and* $x_1\, R_\tau\, y_1$*;*
- $*\, R_1\, *$*;*
- $M(c)\, R_\sigma\, N(c)$ *for every constant* c *in* Σ *of type* σ*.*

The data for a binary logical relation are therefore completely determined by its behaviour on base types. The fundamental result about logical relations underlying all of their applications is the following.

Lemma 2.3 (Basic Lemma for Logical Relations). *Let* R *be a binary logical relation from* M *to* N*; for any term* $t\colon \sigma$ *of* L *in context* Γ*, if* $x\, R_\Gamma\, y$*, then* $M(\Gamma \vdash t\colon \sigma)x\, R_\sigma\, N(\Gamma \vdash t\colon \sigma)y$*,*

where $x\, R_\Gamma\, y$ is an abbreviation for $x_i\, R_{\sigma_i}\, y_i$ for all i where $\sigma_1, \ldots, \sigma_n$ is the sequence of types in Γ. It is routine to define n-ary logical relations for an arbitrary natural number n, in the spirit of Definition 2.2. The corresponding formulation of the Basic Lemma holds for arbitrary n too.

We now outline a *categorical* formulation of logical relations [MR91, MS92]; this will be relaxed slightly to yield our semantic characterisation of algebraic relations for typed lambda calculi with products.

The language L determines a cartesian closed term category, which we also denote by L, such that a model M of the language L in any cartesian closed category such as *Set* extends uniquely to a cartesian closed functor from L to *Set* [Mi96, Sect. 7.2.6]; i.e., a functor that preserves products and exponentials *strictly* (not just up to isomorphism). We may therefore identify the notion of model of the language L in *Set* with that of a cartesian closed functor from L to *Set*.

Definition 2.4. *The category Rel_2 is defined as follows: an object (X, R, Y) consists of a pair of sets X and Y, and a binary relation $R \subseteq X \times Y$; a map from (X, R, Y) to (X', R', Y') is a pair of functions $(f\colon X \to X', g\colon Y \to Y')$ such that $x\,R\,y$ implies $f(x)\,R'\,g(y)$:*

$$\begin{tikzcd} X \arrow[r, "f"] \arrow[d, leftrightarrow, "R"'] & X' \arrow[d, leftrightarrow, "R'"] \\ Y \arrow[r, "g"] & Y' \end{tikzcd}$$

Composition is given by ordinary composition of functions.

We denote the forgetful functors from Rel_2 to Set sending (X, R, Y) to X or to Y by δ_0 and δ_1, respectively.

Proposition 2.5. *Rel_2 is cartesian closed, and the cartesian closed structure is strictly preserved by the functor $(\delta_0, \delta_1)\colon Rel_2 \longrightarrow Set \times Set$.*

For example, $(X_0, R, Y_0) \Rightarrow (X_1, S, Y_1)$ is $([X_0 \Rightarrow X_1], (R \Rightarrow S), [Y_0 \Rightarrow Y_1])$ where $f(R \Rightarrow S)g$ iff, for all $x \in X_0$ and $y \in Y_0$, xRy implies $(fx)S(gy)$.

These properties of Rel_2, combined with the fact that L is freely generated by a signature for cartesian closed categories (i.e., is the generic model on a suitable sketch [KO+97]), are the key to understanding logical relations categorically, as shown by the following.

Proposition 2.6. *To give a binary logical relation from M to N is equivalent to giving a cartesian closed functor $R\colon L \to Rel_2$ such that $(\delta_0, \delta_1)R = (M, N)$:*

$$\begin{tikzcd} & Rel_2 \arrow[d, "(\delta_0, \delta_1)"] \\ L \arrow[ur, "R"] \arrow[r, "(M, N)"'] & Set \times Set \end{tikzcd}$$

Proof. Given a binary logical relation, one immediately has the object function of $R\colon L \to Rel_2$. The equation $(\delta_0, \delta_1)R = (M, N)$ determines the behaviour of R on arrows. The fact that, for any term t of type σ in context Γ, the pair $\big(M(\Gamma \vdash t\colon \sigma), N(\Gamma \vdash t\colon \sigma)\big)$ satisfies the condition making it an arrow in Rel_2 from R_Γ to R_σ follows from (and is equivalent to) the Basic Lemma.

The converse construction is given by taking the object part of a cartesian closed functor $R\colon L \to Rel_2$. It is routine to verify that the two constructions are mutually inverse. ∎

This situation generalises to categories other than Set and Rel_2, the central point being that both categories are cartesian closed and (δ_0, δ_1) is a cartesian closed functor. We outline the following important example which arises in domain theory to deal with logical relations in the context of least fixed points. Let C be the category of ω-cpos with $\perp$ and continuous maps, and M be the class of admissible monos; then there is an evident cartesian closed functor from $Sub_2(C, M)$, the category of admissible binary relations between cpos, to $C \times C$.

A logical relation in this framework is then a cartesian closed functor from L to $Sub_2(C, M)$ (coherent with appropriate models of L in C).

Obviously, one can define a category Rel_n of n-ary relations for an arbitrary natural number n; Propositions 2.5 and 2.6 generalise routinely to arbitrary n.

3 Lax Logical Relations

In this section, we generalise the categorical notion of logical relation to what we call a *lax* logical relation.

Definition 3.1. *Given a signature Σ and the language L generated by Σ, and two models M and N of L in Set, a* (binary) lax logical relation from M to N *is a functor $R\colon L \to Rel_2$ that strictly preserves finite products and satisfies $(\delta_0, \delta_1)R = (M, N)$.*

Note that exponentials are not necessarily preserved. Evidently, one can adapt this definition to one for n-ary lax logical relations for arbitrary n.

The origin of our terminology is as follows. Any finite-product preserving functor $R\colon C \to D$ between cartesian closed categories induces a family of *lax* maps $App_{\sigma,\tau}\colon R_{\sigma\to\tau} \longrightarrow [R_\sigma \Rightarrow R_\tau]$, obtained by taking the Currying in D of the composites

$$R_{\sigma\to\tau} \times R_\sigma \longrightarrow R_{(\sigma\to\tau)\times\sigma} \longrightarrow R_\tau$$

where the first map is determined by preservation of finite products, and the second map is obtained by applying R to the evaluation map in C. This is an instance of (op)lax preservation of structure, specifically, exponential structure.

The notion of Henkin model is closely related to this definition. A Henkin model of the simply typed λ-calculus is a finite-product preserving functor from L to *Set* such that the induced lax maps are injective. This is a kind of lax model, but is not quite the same as giving a unary lax logical relation; nevertheless, it is a natural and useful generalisation of the notion of model we have used, and one to which our results routinely extend.

The Basic Lemma for logical relations extends to lax logical relations; in fact, the lax logical relations can be characterised in terms of a Basic Lemma.

Lemma 3.2 (Basic Lemma for Lax Logical Relations). *Let M and N be models of L in Set. A family of relations $R_\sigma \subseteq M_\sigma \times N_\sigma$ for every type σ of L determines a lax logical relation from M to N if and only if, for every term $t\colon\sigma$ of L in context Γ, if $x\, R_\Gamma\, y$, then $M(\Gamma \vdash t\colon\sigma)x\, R_\sigma\, N(\Gamma \vdash t\colon\sigma)y$,*

where $x\, R_\Gamma\, y$ is an abbreviation for $x_i\, R_{\sigma_i}\, y_i$ for all i when $\sigma_1, \ldots, \sigma_n$ is the sequence of types in Γ.

Proof. For the forward (only-if) direction, suppose Γ has sequence of types $\sigma_1, \ldots, \sigma_n$. The expression $\Gamma \vdash t\colon\sigma$ is a map in L from $\sigma_1 \times \cdots \times \sigma_n$ to σ, so R sends it to the unique map from $R_{\sigma_1\times\cdots\times\sigma_n}$ to R_σ in Rel_2 that lifts the pair $\big(M(\Gamma \vdash t\colon\sigma), N(\Gamma \vdash t\colon\sigma)\big)$:

$$\begin{array}{ccc} M(\sigma_1 \times \cdots \times \sigma_n) & \xrightarrow{M(\Gamma \vdash t:\sigma)} & M(\sigma) \\ R_{\sigma_1 \times \cdots \times \sigma_n} \updownarrow & & \updownarrow R_\sigma \\ N(\sigma_1 \times \cdots \times \sigma_n) & \xrightarrow{N(\Gamma \vdash t:\sigma)} & N(\sigma) \end{array}$$

If $x_i \, R_{\sigma_i} \, y_i$ for all i then $(x_1, \ldots, x_n) \, R_{\sigma_1 \times \cdots \times \sigma_n} \, (y_1, \ldots, y_n)$ because R preserves finite products, and so the result is now immediate, as $x \in M(\sigma_1 \times \cdots \times \sigma_n) = (x_1, \ldots, x_n) \in M(\sigma_1) \times \cdots \times M(\sigma_n)$ and similarly for y.

For the converse, first taking Γ to be a singleton, the condition uniquely determines maps $R(\Gamma \vdash t:\sigma): R(\Gamma) \to R(\sigma)$ in Rel_2, giving a graph morphism from L to Set such that $(\delta_0, \delta_1)R = (M, N)$. Such a graph morphism is trivially necessarily a functor. Taking $\Gamma \vdash t:\sigma$ to be $\phi \vdash *:1$, where $*$ is the unique constant of type 1, the condition yields $* \, R_1 \, *$, so R preserves the terminal object. Taking $\Gamma \vdash t:\sigma$ to be $a:\sigma_0, b:\sigma_1 \vdash (a,b):\sigma_0 \times \sigma_1$ yields that if $x_0 \, R_{\sigma_0} \, y_0$ and $x_1 \, R_{\sigma_1} \, y_1$, then $(x_0, x_1) \, R_{\sigma_0 \times \sigma_1} \, (y_0, y_1)$. And taking $\Gamma \vdash t:\sigma$ to be $a:\sigma_0 \times \sigma_1 \vdash \pi_i \, a:\sigma_i$ for $i = 0, 1$ give the converse. So R preserves finite products. ■

We conclude this section by showing how lax logical relations can be used for the two applications of [HS99] previously discussed.

Definition 3.3. *If R is a type-indexed family of binary relations from M to N and S is a type-indexed family of binary relations from N to P, their* composite *$R \,;\, S$ is defined component-wise; i.e., $(R \,;\, S)_\sigma = R_\sigma \,;\, S_\sigma$*

where ; on the right-hand side denotes the conventional composition of binary relations.

Proposition 3.4. *If R is a binary lax logical relation from M to N and S is a binary lax logical relation from N to P, then $R \,;\, S$ is a lax logical relation from M to P.*

Proof. We must show that if $R: L \to Rel_2$ and $S: L \to Rel_2$ strictly preserve finite products, then so does $R \,;\, S$. But $(x_0, x_1) \, (R \,;\, S)_{\sigma \times \tau} \, (y_0, y_1)$ if and only if there exists (z_0, z_1) such that $(x_0, x_1) \, R_{\sigma \times \tau} \, (z_0, z_1)$ and $(z_0, z_1) \, S_{\sigma \times \tau} \, (y_0, y_1)$, and that is so if and only if $x_0 \, (R \,;\, S)_\sigma \, y_0$ and $x_1 \, (R \,;\, S)_\tau \, y_1$. The proof for a terminal object is trivial. ■

Various other closure properties (such as closure with respect to conjunction and universal and existential quantification) have been proved in [HS99] for pre-logical relations; the results in the following section show that lax logical relations also have these closure properties.

Definition 3.5. *Let M and N be models of L in Set, and OBS be a set of types; then M and N are said to be* observationally equivalent with respect to *OBS (written $M \equiv_{OBS} N$) when, for all $\sigma \in OBS$ and all closed $t, t':\sigma$, $M(t) = M(t')$ if and only if $N(t) = N(t')$.*

Proposition 3.6. *$M \equiv_{OBS} N$ if and only if there exists a lax logical relation from M to N which is one-to-one for every $\sigma \in OBS$.*

Proof. For the forward direction, consider the family of relations $R_\sigma \subseteq M_\sigma \times N_\sigma$ defined by $aR_\sigma b$ if and only if there exists a closed term $t\colon \sigma$ such that $M(t) = a$ and $N(t) = b$. This is one-to-one on observable types because of observational equivalence and a lax logical relation because $R_{\sigma\times\tau} = R_\sigma \times R_\tau$.

For the converse, suppose $R_\sigma \subseteq M_\sigma \times N_\sigma$ determine a lax logical relation; if $\sigma \in OBS$ then, for all closed $t\colon\sigma$, $M(t)\, R_\sigma\, N(t)$ by the Basic Lemma and $M(t) = M(t')$ if and only if $N(t) = N(t')$ because R_σ is one-to-one. ■

4 Pre-logical Relations

We can use the Basic Lemma of Sect. 3 and the corresponding result of [HS99] to see immediately that, for models as we defined them in Sect. 2, the notions of lax logical relation and pre-logical relation coincide. However, in this section we give a more direct exposition of the connection for a larger class of models. In [HS99], the analysis is primarily in terms of the simply typed λ-calculus without product types. But they mention the case of λ-calculi with products and models that satisfy surjective pairing. Hence, consider models now to be functors $M\colon L \to Set$ that strictly preserve finite products (but not necessarily exponentials); these include Henkin models. Everything we have said about lax logical relations extends routinely to this class of models.

Definition 4.1. *A* pre-logical relation from M to N *consists of, for each type* σ*, a relation* $R_\sigma \subseteq M_\sigma \times N_\sigma$ *such that*

1. *if* $x\, R_\sigma\, y$ *and* $f\, R_{\sigma\to\tau}\, g$*, then* $App_{\sigma,\tau} f x\, R_\tau\, App_{\sigma,\tau} g y$*, where maps* $App_{\sigma,\tau}$ *are determined by finite-product preservation of* M *and* N*, respectively, as discussed in Sect. 3;*
2. $M(c)\, R_\sigma\, N(c)$ *for every constant* c *of type* σ*, where the constants are deemed to include*
   - *all constants in* Σ*,*
   - $*\colon 1$*,*
   - $(-,-)\colon \sigma \to \tau \to (\sigma \times \tau)$*,*
   - $\pi_0 : \sigma \times \tau \to \sigma$ *and* $\pi_1 : \sigma \times \tau \to \tau$*, and*
   - *all instances of combinators* $S_{\rho,\sigma,\tau}\colon (\rho \to \sigma \to \tau) \to (\rho \to \sigma) \to \rho \to \tau$ *and* $K_{\sigma,\tau}\colon \sigma \to \tau \to \sigma$*.*

Theorem 4.2. *A type-indexed family of relations* $R_\sigma \subseteq M_\sigma \times N_\sigma$ *determines a lax logical relation from* M *to* N *if and only if it is a pre-logical relation from* M *to* N*.*

Proof. For the second clause in the forward direction, treat all constants as maps in L with domain 1. For the first clause, note that $R_\sigma \times R_{\sigma\to\tau} = R_{\sigma\times(\sigma\to\tau)}$, so applying functoriality of R to the evaluation map $ev\colon \sigma \times (\sigma \to \tau) \longrightarrow \tau$ in L, the result follows immediately.

For the converse, the second condition implies that, for all closed terms t, $M(t)\, R\, N(t)$; that fact, combined with the fact that every map in L is an un-Currying of a closed term, plus the first condition, imply that R is a graph

morphism making $(\delta_0, \delta_1)R = (M, N)$, hence trivially a functor. Since $*\colon 1$ is a constant and $M(*) = * = N(*)$, we have $M(*)\, R_1\, N(*)$; so R preserves the terminal object. Since $(-,-)$ is a constant, it follows that if $x_0\, R_{\sigma_0}\, y_0$ and $x_1\, R_{\sigma_1}\, y_1$, then $(x_0, x_1)\, R_{\sigma_0 \times \sigma_1}\, (y_0, y_1)$. The inverse holds because π_0 and π_1 are maps in L. So R preserves finite products. ■

5 Another Syntax-Based Characterisation

The key point in the pre-logical characterisation above is that every map in the category L is generated by the constants. In this section, we give an alternative syntax-based characterisation that generalizes more directly to other languages. For simplicity of exposition, we assume, as previously, that models preserve exponentials as well as products.

Theorem 5.1. *To give a lax logical relation from M to N is equivalent to giving, for each type σ of L, a relation $R_\sigma \subseteq M_\sigma \times N_\sigma$ such that*

1. *if $f\, R_{(\sigma\times\tau)\to\rho}\, g$, then $Curry(f)\, R_{\sigma\to\tau\to\rho}\, Curry(g)$*
2. *$App\, R_{((\sigma\to\tau\times\sigma)\to\tau}\, App$*
3. *if $f_0\, R_{\sigma\to\tau}\, g_0$ and $f_1\, R_{\sigma\to\rho}\, g_1$, then $(f_0, f_1)\, R_{\sigma\to(\tau\times\rho)}\, (g_0, g_1)$*
4. *$\pi_0\, R_{\sigma\times\tau\to\sigma}\pi_0$ and $\pi_1\, R_{\sigma\times\tau\to\tau}\, \pi_1$*
5. *if $f\, R_{\sigma\to\tau}\, g$ and $f'\, R_{\tau\to\rho}\, g'$, then $(f' \cdot f)R_{\sigma\to\rho}(g' \cdot g)$*
6. *$id\, R_{\sigma\to\sigma}\, id$*
7. *$x\, R_\sigma\, y$ if and only if $x\, R_{1\to\sigma}\, y$*
8. *$M(c)\, R_\sigma\, N(c)$ for every base term c in Σ of type σ.*

We chose the conditions above because the first four conditions seem particularly natural from the perspective of the λ-calculus, the following two, which are about substitution, are natural category theoretic conditions, the seventh is mundane, and the last evident; *cf.* the "categorical combinators" of [Cu93].

Proof. For the forward direction, the relations R_σ are given by the object part of the functor. The conditions follow immediately from the fact of R being a functor, thereby having an action on all maps, and from the fact that it strictly preserves finite products. For instance, there is a map in L from $(\sigma \to \tau) \times (\sigma \to \rho)$ to $\sigma \to (\tau \times \rho)$, so that map is sent by R to a map in Rel_2, and R strictly preserves finite products, yielding the third condition. So using the definition of a map in Rel_2, and the facts that $(\delta_0, \delta_1)R = (M, N)$ and that M and N are strict structure preserving functors, we have the result.

For the converse, the family of relations gives the object part of the functor R. Observe that the axioms imply

- $(x_0, x_1)\, R_{\sigma\times\tau}\, (y_0, y_1)$ if and only if $x_0\, R_\sigma\, y_0$ and $x_1\, R_\tau\, y_1$
- $*\, R_1\, *$, where $*$ is the unique element of $M_1 = N_1 = 1$

So, R strictly preserves finite products providing it forms a functor. The data for M and N and the desired coherence condition $(\delta_0, \delta_1)R = (M, N)$ on the putative functor determine its behaviour on maps. It remains to check that the

image of every map in L actually lies in Rel_2. But the conditions inductively define the Currying of every map in L, so unCurrying by the fifth and seventh conditions, the result follows. It is routine to verify that these constructions are mutually inverse. ■

The result holds for the more general class of models we have discussed, but an exposition would be encumbered by numerous occurrences of $App_{\sigma,\tau}$. It is routine to generalise Theorems 4.2 and 5.1 to n-ary relations for arbitrary n.

6 Models in Cartesian Closed Categories

Cartesian closed categories are a more general class of models for typed lambda calculi. In this section, we consider a model to be a functor from L to a cartesian closed category, strictly preserving finite products and exponentials.

To discuss "relations" in this context, we adopt the sub-scone approach described in [La88, MR91, MS92, Al95]. Let C be a cartesian closed category, S be a finitely complete cartesian closed category, and $G\colon C \to S$ be a functor that preserves products (up to isomorphism). A typical example of a suitable functor G is $\hom(1,-)\colon C \to Set$, the global-elements functor; other examples may be found in the references given above. Then these data determine a category $G\text{-}Rel_2$ of *categorical (binary) relations on* C as follows.

Let $Rel_2(S)$ be the category of binary relations on S with evident forgetful functor $Rel_2(S) \longrightarrow S \times S$; then pulling back along $G \times G$ determines a category $G\text{-}Rel_2$ and a forgetful functor to $C \times C$. In detail, the objects of $G\text{-}Rel_2$ are triples (a_0, s, a_1) where a_0 and a_1 are objects of C and s is a sub-object of $G(a_0) \times G(a_1)$; the morphisms from (a_0, s, a_1) to (b_0, t, b_1) are triples (f_0, q, f_1) such that $f_i\colon a_i \to b_i$ in C, $q\colon \operatorname{dom} s \to \operatorname{dom} t$ in S, and the following diagram commutes:

$$\begin{tikzcd}
\cdot \arrow[r, tail, "s"] \arrow[d, "q"'] & G(a_0) \times G(a_1) \arrow[d, "G(f_0) \times G(f_1)"] \\
\cdot \arrow[r, tail, "t"] & G(b_0) \times G(b_1)
\end{tikzcd}$$

Composition and identities are evident. The forgetful functors $\delta_i\colon G\text{-}Rel_2 \to C$ for $i = 0, 1$ are defined by $\delta_i(a_0, s, a_1) = a_i$ and similarly for morphisms.

Proposition 6.1. *$G\text{-}Rel_2$ is a cartesian closed category and the cartesian closed structure is strictly preserved by $(\delta_0, \delta_1)\colon G\text{-}Rel_2 \longrightarrow C \times C$; furthermore, this functor is faithful.*

Definition 6.2. *Given a signature Σ and the language L generated by Σ, two models M and N of L in a cartesian closed category C, and a category $G\text{-}Rel_2$ of binary categorical relations on C, a* (binary) lax logical relation from M to N *is a functor $R\colon L \longrightarrow G\text{-}Rel_2$ that satisfies $(\delta_0, \delta_1)R = (M, N)$ and strictly preserves finite products.*

Lemma 6.3 (Basic Lemma for Categorical Lax Logical Relations). *Let M and N be models of L in a cartesian closed category C, S be a finitely complete cartesian closed category, and $G\colon C \to S$ preserve finite products up to isomorphism; then a family of sub-objects $R_\sigma\colon \cdot \rightarrowtail G(M_\sigma) \times G(N_\sigma)$ for every type σ of L determines a lax logical relation from M to N if and only if, for every term t of L of type σ in context Γ, there exists a unique map q that makes the following diagram commute:*

$$\begin{tikzcd}
\cdot \arrow[r, tail, "\Pi_i R_{\sigma_i}"] \arrow[d, dotted, "q"'] & G(\Pi_i M_{\sigma_i}) \times G(\Pi_i N_{\sigma_i}) \arrow[d, "G(M(t)) \times G(N(t))"] \\
\cdot \arrow[r, tail, "R_\sigma"] & G(M_\sigma) \times G(N_\sigma)
\end{tikzcd}$$

where $\sigma_1, \ldots, \sigma_n$ is the sequence of types in Γ.

Proof. For the forward direction, R maps $\Gamma \vdash t\colon \sigma$ to a map

$$\begin{tikzcd}
\cdot \arrow[r, tail, "R_{\Pi_i \sigma_i}"] \arrow[d, "q"'] & G(M_{\Pi_i \sigma_i}) \times G(M_{\Pi_i \sigma_i}) \arrow[d, "G(M(t)) \times G(N(t))"] \\
\cdot \arrow[r, tail, "R_\sigma"] & G(M_\sigma) \times G(N_\sigma)
\end{tikzcd}$$

The result follows because R, M and N preserve products.

In the converse direction, the morphism part of the functor is determined by the assumed maps. Taking $\Gamma \vdash t\colon \sigma$ to be $a\colon \sigma_0, b\colon \sigma_1 \vdash (a, b)\colon \sigma_0 \times \sigma_1$ shows that $R_{\sigma_0} \times R_{\sigma_1} \leq R_{\sigma_0 \times \sigma_1}$, using the fact that G, M and N all preserve products, and taking $\Gamma \vdash t\colon \sigma$ to be $p\colon \sigma_0 \times \sigma_1 \vdash \pi_i p\colon \sigma_i$ for $i = 0, 1$ shows the converse. Finally, taking $\Gamma \vdash t\colon \sigma$ to be $\emptyset \vdash *\colon 1$ shows that R_1 is the "true" sub-object of $G(M_1) \times G(N_1)$. So R preserves products. ■

This result can be generalised: replace $G\text{-}Rel_2$ and (δ_0, δ_1) by any category D with finite products and a faithful finite-product preserving functor to $C \times C$. This would amount to a lax version of Peter Freyd's suggestion [Mi90, Section 3.6.4] of studying logical relations as subcategories that respect cartesian-closed (here, cartesian) structure, except generalised from subcategory to faithful functor. But many applications require entailments to, or from, the "relations," and so a lax version of Hermida's [He93] fibrations with structure to support a $(\top, \wedge, \Rightarrow, \forall)$ logic might be a more appropriate level of generality.

To consider composition of (binary) lax logical relations in this context, assume first that S is the usual category of sets and functions; then the objects of $G\text{-}Rel_2$ are subsets of sets of the form $G(a) \times G(b)$.

Proposition 6.4. *Composition of (binary) categorical lax logical relations can be defined component-wise.*

To allow recursion in L, consider again the category $Sub_2(C, M)$ discussed at the end of Section 2, with C being the category of ω-cpos with $\perp$ and M being the admissible monos. Using the sconing functor $G = C(1, -)\colon C \to Set$ gives us a category $G\text{-}Rel_2$ as above; because this is constructed as a pullback, there

exists a strict finite-product preserving functor F from $Sub_2(C, M)$ to $G\text{-}Rel_2$. Given any logical relation functor $R\colon L \to Sub_2(C, M)$, composing with F gives a strict finite-product preserving functor from L to $G\text{-}Rel_2$ (i.e., a lax logical relation) between the original models. This shows how composition is supported in the context of relations on ω-cpos.

More generally, if S is assumed to be a *regular* category [Bo94], a relational composition can be defined. Any pre-sheaf category Set^W, or indeed any topos, is a regular category, so this is a mild assumption. An *axiomatic* treatment of composition of generalized logical relations, including lax logical relations as discussed here, can be found in [KO$^+$97], which emerged from category theoretic treatments of data refinement in which composition of refinements is crucial [JH90, KP96, KP].

7 Generalising from the λ-Calculus

In Sect. 3, the fundamental facts that gave rise to our definition of lax logical relation were the correspondence between the simply typed λ-calculus and cartesian closed categories, and the fact that a signature Σ gave rise to a cartesian closed category L such that a model of Σ could be seen as a functor from L into *Set* (or, more generally, any cartesian closed category) that strictly preserved cartesian closed structure. So in generalising from the simply typed λ-calculus, we generalise the latter fact. This may be done in terms of algebraic structure, or equivalently (finitary) monads, on *Cat*. The central paper about that is Blackwell, Kelly and Power's [BKP89]. We can avoid much of the subtlety here by restricting our attention to maps that preserve structure strictly.

We shall first describe the situation for an arbitrary (finitary) monad T on *Cat* extending finite-product structure. One requires *Set* (or, more generally, any small category C) to have T-structure, L to be the free T-algebra generated by a signature, and define a model M of L to be a strict T-algebra map, *cf.* [KO$^+$97].

A natural general setting in which to define the notion of lax logical relation involves assuming the existence of a small category E (with finite products) of relations, and a strict finite-product preserving forgetful functor (δ_0, δ_1) from E to $C \times C$. One then adds to these data further categorical structure inside the category of small categories and functors that strictly preserve finite products to generalise the composition of binary relations. These definitions and related results appear in [KO$^+$97]. Here, we aim to state a Basic Lemma in familiar terms, and so restrict attention to the special case that $C = Set$ and $E = Rel_2$.

A *lax logical relation* is a strict finite-product preserving functor from L into Rel_2 such that composition with (δ_0, δ_1) yields (M, N).

We can generalise the Basic Lemma to this level of generality as follows.

Lemma 7.1 (Basic Lemma for Lax Logical Relations with Algebraic Structure). *A family of relations $R_\sigma \subseteq M_\sigma \times N_\sigma$ for every type σ of L determines a lax logical relation from M to N if and only if, for every term t of L of type σ in context Γ, if $x\, R_\Gamma\, y$, then $M(\Gamma \vdash t\colon \sigma)x\, R_\sigma\, N(\Gamma \vdash t\colon \sigma)y$*

The proof is exactly as in Sect. 3; similarly,

Proposition 7.2. *Binary lax logical relations (at the current level of generality) compose component-wise.*

In the above we have tacitly assumed that contexts are modelled by finite products. In general, there is no need to make this assumption: contexts could be modelled by a symmetric monoidal structure or, more generally, by a symmetric pre-monoidal structure, or Freyd structure. It would be straightforward to generalise our analysis to include such possibilities, but it may be simpler to deal with them case by case. For an analysis of the notion of lax logical relations where contexts are modelled by a Freyd structure, see [KP99].

We next want to generalise Theorem 5.1. In order to do that, we need to consider the formulation of finitary monads in terms of algebraic structure on *Cat*, and we need to restrict to a particular class of such structures. The general notion of algebraic structure, and the relevant results, appear in [KP93] and [Po97], and we have included it in the Appendix. Using the notation of the Appendix, we consider a special class of algebraic structure.

Definition 7.3. *Algebraic structure* (S, E) *on Cat is* discrete *if* $S(c) = 0$ *whenever* c *is not a discrete category, i.e., whenever* c *is not the discrete category on a finite set.*

It follows from the definition that any discrete algebraic structure may be presented by two families of operations: *object* operations, which have algebras given by functors of the form $C^k \longrightarrow C$, and *arrow* operations, which are given by natural transformations between object operations. One may put equations between these to obtain all operations of any discrete algebraic structure, which are given by functors $C^k \longrightarrow C^{Sk}$, where Sk is a small category.

Assuming *Set* has (S, E)-structure for some given discrete algebraic structure (S, E), a *model* of an (S, E)-algebra in *Set* is a functor that strictly preserves (S, E)-structure.

Examples of discrete algebraic structure have models given by small categories with finite products, with finite coproducts, with monoidal structure, symmetric monoidal structure, a monad [Mo91], an endofunctor, a natural transformation between endofunctors, or any combination of the above.

In order to extend Theorem 5.1, rather than give an analogue, we must include exponentials, although they are not instances of discrete algebraic structure as we have defined it. So we henceforth assume that we are given discrete algebraic structure on *Cat* extending finite-product structure; that L is generated by the simply typed λ-calculus, a signature, and the discrete algebraic structure; that M and N are models of L in *Set* strictly preserving the algebraic structure, and, restricting our definition above, that a *lax logical relation* from M to N is a finite-product preserving functor from L to Rel_2 such that composition with (δ_0, δ_1) yields (M, N).

A methodology for extending Theorem 5.1 is as follows. Algebraic structure on *Cat* is given by an equational presentation. That equational presentation has operations defining objects and arrows. For each operation defining an arrow, one adds an axiom to the list in the statement of Theorem 5.1 in the same spirit. For

instance, to define a monoidal structure, one has operations that assign to each pair of maps (f, g), a map $f \otimes g$, and gives associative maps and their inverses, and left and right unit maps and their inverses. So one would add axioms

- if $f_0 \, R_{\sigma_0 \to \tau_0} \, g_0$ and $f_1 \, R_{\sigma_1 \to \tau_1} \, g_1$ then $(f_0 \otimes f_1) \, R_{(\sigma_0 \otimes \sigma_1) \to (\tau_0 \otimes \tau_1)} \, (g_0 \otimes g_1)$;
- $a \, R_{(\sigma \otimes \tau) \otimes \rho \to \sigma \otimes (\tau \otimes \rho)} \, a$, $l \, R_{(\sigma \otimes I) \to \sigma} \, l$ and $r \, R_{(I \otimes \sigma) \to \sigma} \, r$;
- $a^{-1} \, R_{\sigma \otimes (\tau \otimes \rho) \to (\sigma \otimes \tau) \otimes \rho} \, a^{-1}$, $l^{-1} \, R_{\sigma \to (\sigma \otimes I)} \, l^{-1}$ and $r^{-1} \, R_{\sigma \to (I \otimes \sigma)} \, r^{-1}$.

Theorem 7.4. *For any discrete algebraic structure on Cat extending finite-product structure, to give a lax logical relation from M to N is equivalent to giving a family of relations $R_\sigma \subseteq M_\sigma \times N_\sigma$ satisfying the conditions of Theorem 5.1 and also*

- *for any k-ary object operation O, if $f_i \, R_{\sigma_i \to \tau_i} \, g_i$ for all $1 \leq i \leq k$, then*

$$O(f_1, \ldots, f_k) \, R_{O(\sigma_1, \ldots, \sigma_k) \to O(\tau_1, \ldots, \tau_k)} \, O(g_1, \ldots, g_k)$$

- *for any k-ary arrow operation O, we have*

$$O(M\sigma_1, \ldots, M\sigma_k) \, R_\gamma \, O(N\sigma_1, \ldots, N\sigma_k)$$

  where $\gamma = \operatorname{dom} O(\sigma_1, \ldots, \sigma_k) \longrightarrow \operatorname{cod} O(\sigma_1, \ldots, \sigma_k)$
- *for any k-ary arrow operation O, if $f_i \, R_{\sigma_i \to \tau_i} \, g_i$ for all $1 \leq i \leq k$, then*

$$\operatorname{dom} O(f_1, \ldots, f_k) \, R_\beta \, \operatorname{dom} O(g_1, \ldots, g_k)$$

  where $\beta = \operatorname{dom} O(\sigma_1, \ldots, \sigma_k) \longrightarrow \operatorname{dom} O(\tau_1, \ldots, \tau_k)$, *and similarly with* dom *systematically replaced by* cod.

The final two rules here may seem unfamiliar at first sight. An arrow operation takes a k-ary family of objects to an arrow, so syntactically, takes a k-ary family of types to an equivalence class of terms. That leads to our penultimate rule. That a k-ary arrow operation is functorial means that every k-ary family of arrows is sent to a commutative square. So we need rules to the effect that every arrow in that square behaves as required. The penultimate rule above accounts for two arrows of the square, and the final rule accounts for the other two, where the domain of the commutative square is the arrow of the square uniquely determined by the definitions.

Proof. Follow the proof of Theorem 5.1. The conditions show inductively that for every arrow of the category freely generated by the given discrete algebraic structure applied to the signature, one obtains an arrow in *Rel*. ■

Note that this allows for dropping exponentials, as well as adding various kinds of structure such as finite co-products and tensor products. We hope this generality will lead to interesting new applications.

References

[Ab90] S. Abramsky. Abstract interpretation, logical relations and Kan extensions. *J. of Logic and Computation*, 1:5–40, 1990.

[Al95] M. Alimohamed. A characterization of lambda definability in categorical models of implicit polymorphism. *Theoretical Computer Science*, 146:5–23, 1995.

[BKP89] R. Blackwell, H. M. Kelly, and A. J. Power. Two dimensional monad theory. *J. of Pure and Applied Algebra*, 59:1–41, 1989.

[Bo94] Francis Borceux. *Handbook of Categorical Algebra 2*, volume 51 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1994.

[Cu93] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming*. Birkhauser, Boston, 1993.

[FRA99] J. Flum and M. Rodriguez-Artalejo, editors. *Computer Science Logic, 13th International Workshop, CSL'99*, volume 1683 of *Lecture Notes in Computer Science*, Madrid, Spain, September 1999. Springer-Verlag, Berlin (1999).

[Gi68] A. Ginzburg. *Algebraic Theory of Automata*. Academic Press, 1968.

[He93] Claudio A. Hermida. *Fibrations, logical predicates, and indeterminates*. Ph.D. thesis, The University of Edinburgh, 1993. Available as Computer Science Report CST–103–93 or ECS–LFCS–93–277.

[HL+] F. Honsell, J. Longley, D. Sannella, and A. Tarlecki. Constructive data refinement in typed lambda calculus. To appear in the Proceedings of FOSSACS 2000, Springer-Verlag *Lecture Notes in Computer Science*.

[HS99] F. Honsell and D. Sannella. Pre-logical relations. In Flum and Rodriguez-Artalejo [FRA99], pages 546–561.

[JH90] He Jifeng and C. A. R. Hoare. Data refinement in a categorical setting. Technical monograph PRG-90, Oxford University Computing Laboratory, Programming Research Group, Oxford, November 1990.

[JT93] A. Jung and J. Tiuryn. A new characterization of lambda definability. In M. Bezen and J. F. Groote, editors, *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, pages 245–257, Utrecht, The Netherlands, March 1993. Springer-Verlag, Berlin.

[KO+97] Y. Kinoshita, P. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. An axiomatic approach to binary logical relations with applications to data refinement. In M. Abadi and T. Ito, editors, *Theoretical Aspects of Computer Software*, volume 1281 of *Lecture Notes in Computer Science*, pages 191–212, Sendai, Japan, 1997. Springer-Verlag, Berlin.

[KP] Y. Kinoshita and A. J. Power. Data refinement by enrichment of algebraic structure. To appear in *Acta Informatica*.

[KP93] G. M. Kelly and A. J. Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *Journal of Pure and Applied Algebra*, 89:163–179, 1993.

[KP96] Y. Kinoshita and A. J. Power. Lax naturality through enrichment. *J. Pure and Applied Algebra*, 112:53–72, 1996.

[KP99] Y. Kinoshita and J. Power. Data refinement for call-by-value programming languages. In Flum and Rodriguez-Artalejo [FRA99], pages 562–576.

[La88] Y. Lafont. *Logiques, Categories et Machines*. Thèse de Doctorat, Université de Paris VII, 1988.

[Mi71] R. Milner. An algebraic definition of simulation between programs. In *Proceedings of the Second International Joint Conference on Artificial Intelligence*, pages 481–489. The British Computer Society, London, 1971. Also Technical Report CS-205, Computer Science Department, Stanford University, February 1971.

[Mi90] J. C. Mitchell. Type systems for programming languages. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 365–458. Elsevier, Amsterdam, and The MIT Press, Cambridge, Mass., 1990.

[Mi91] J. C. Mitchell. On the equivalence of data representations. In V. Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 305–330. Academic Press, 1991.

[Mi96] J. C. Mitchell. *Foundations for Programming Languages.* The MIT Press, 1996.

[Mo91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, July 1991.

[MR91] QingMing Ma and J. C. Reynolds. Types, abstraction, and parametric polymorphism, part 2. In S. Brookes, M. Main, A. Melton, M. Mislove, and D. Schmidt, editors, *Mathematical Foundations of Programming Semantics, Proceedings of the 7th International Conference*, volume 598 of *Lecture Notes in Computer Science*, pages 1–40, Pittsburgh, PA, March 1991. Springer-Verlag, Berlin (1992).

[MS76] R. E. Milne and C. Strachey. *A Theory of Programming Language Semantics.* Chapman and Hall, London, and Wiley, New York, 1976.

[MS92] J. C. Mitchell and A. Scedrov. Notes on sconing and relators. In E. Börger, G. Jäger, H. Kleine Büning, S. Martini, and M. M. Richter, editors, *Computer Science Logic: 6th Workshop, CSL '92: Selected Papers*, volume 702 of *Lecture Notes in Computer Science*, pages 352–378, San Miniato, Italy, 1992. Springer-Verlag, Berlin (1993).

[OR95] P. O'Hearn and J. Riecke. Kripke logical relations and PCF. *Information and Computation*, 120(1):107–116, 1995.

[OT95] P. W. O'Hearn and R. D. Tennent. Parametricity and local variables. *J. ACM*, 42(3):658–709, May 1995.

[Pl73] G. D. Plotkin. Lambda-definability and logical relations. Memorandum SAI-RM-4, School of Artificial Intelligence, University of Edinburgh, October 1973.

[Pl80] G. D. Plotkin. Lambda-definability in the full type hierarchy. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 363–373. Academic Press, 1980.

[Po97] A. J. Power. Categories with algebraic structure. In M. Nielsen and W. Thomas, editors, *Computer Science Logic, 11th International Workshop, CSL'99*, volume 1414 of *Lecture Notes in Computer Science*, pages 389–405, Aarhus, Denmark, August 1997. Springer-Verlag, Berlin (1998).

[Re74] J. C. Reynolds. On the relation between direct and continuation semantics. In J. Loeckx, editor, *Proc. 2nd Int. Colloq. on Automata, Languages and Programming*, volume 14 of *Lecture Notes in Computer Science*, pages 141–156. Springer-Verlag, Berlin, 1974.

[Re83] J. C. Reynolds. Types, abstraction and parametric polymorphism. In R. E. A. Mason, editor, *Information Processing 83*, pages 513–523, Paris, France, 1983. North-Holland, Amsterdam.

[Sc87] O. Schoett. *Data abstraction and the correctness of modular programming.* Ph.D. thesis, University of Edinburgh, February 1987. Report CST-42-87.

[St96] I. Stark. Categorical models for local names. Lisp *and Symbolic Computation*, 9(1):77–107, February 1996.

Appendix: Algebraic Structure on Categories

In ordinary universal algebra, an algebra is a set X together with a family of basic operations $\sigma\colon X^n \to X$, subject to equations between derived operations. In order to define algebraic structure on categories, one must replace the set X by a category A. One also replaces the finite number n by a finitely presentable category c. All finite categories are finitely presentable, and finite categories are the only finitely presentable categories we need in this paper. One also allows not only functions from the set $Cat(c, A)$ into the set of objects of A, but also functions from the set $Cat(c, A)$ into the set of arrows in A. These are subject to equations between derived operations. It follows that the category of small such categories with structure and functors that strictly preserve the structure is equivalent to the category of algebras, T-Alg, for a finitary monad T on Cat.

All structures relevant to this paper are instances of a slightly more restricted situation: that of Cat-enriched algebraic structure. So we shall restrict to Cat-enriched structures here. Let C denote the 2-category Cat of small categories. So $C(A, B)$ denotes the category of functors from A to B. Let C_f denote the full sub-2-category of C given by (isomorphism classes of) finitely presentable categories.

Definition A.1. *A* signature *on C is a 2-functor $S\colon \mathrm{ob}\, C_f \longrightarrow C$, regarding* $\mathrm{ob}\, C_f$ *as a discrete 2-category.*

For each $c \in \mathrm{ob}\, C_f$, $S(c)$ is called the category of *basic operations of arity c*. Using S, we construct $S_\omega\colon C_f \longrightarrow C$ as follows: set

$$S_0 = J, \text{ the inclusion of } C_f \text{ into } C, \text{ and}$$
$$S_{n+1} = J + \textstyle\sum_{d \in \mathrm{ob}\, C_f} C\big(d, S_n(-)\big) \times S(d);$$

and define

$$\sigma_0\colon S_0 \to S_1 \text{ to be } inj\colon J \longrightarrow\ J + \textstyle\sum_{d \in \mathrm{ob}\, C_f} C\big(d, S_0(-)\big) \times S(d); \text{ and}$$
$$\sigma_{n+1}\colon S_{n+1} \to S_{n+2} \text{ to be } J + \textstyle\sum_{d \in \mathrm{ob}\, C_f} C\big(d, \sigma_n(-)\big) \times S(d).$$

Then $S_\omega = \mathrm{colim}_{n<\omega}\, S_n$, where the colimit exists because C is cocomplete, and it is a colimit in a functor category with base C. In many cases of interest, each σ_n is a monomorphism, so S_ω is the union of $\{S_n\}_{n<\omega}$. For each c, we call $S_\omega(c)$ the category of *derived c-ary operations*.

A signature is typically accompanied by equations between derived operations. So we say

Definition A.2. *The* equations *of an algebraic theory with signature S are given by a 2-functor $E\colon \mathrm{ob}\, C_f \longrightarrow C$ together with 2-natural transformations $\tau_1, \tau_2\colon E \longrightarrow S_\omega\big(K(-)\big)$, where $K\colon \mathrm{ob}\, C_f \longrightarrow C_f$ is the inclusion.*

Definition A.3. Algebraic structure *on C consists of a signature S, together with equations (E, τ_1, τ_2).*

We generally denote algebraic structure by (S, E), suppressing τ_1 and τ_2.

We now define the algebras for a given algebraic structure.

Definition A.4. *Given a signature S, an S-*algebra *consists of a small category A together with a functor $\nu_c\colon C(c, A) \longrightarrow C\big(S(c), A\big)$ for each c.*

So, an S-algebra consists of a carrier A and an interpretation of the basic operations of the signature. This interpretation extends canonically to the derived operations, giving an $S_\omega\big(K(-)\big)$-algebra, as follows.

- $\nu_0\colon C(c, A) \longrightarrow C\big(S_0(c), A\big)$ is the identity;
- using the fact that $C(-, A)$ preserves colimits, to give a functor ν_{n+1} from $C(c, A)$ to $C\big(S_{n+1}(c), A\big)$ is equivalent to giving a functor from $C(c, A)$ to $C(c, A)$, which we will make the identity, and, for each d in $\operatorname{ob} C_f$, a functor from $C(c, A)$ to $C\big(C(d, S_n(c)\big), C\big(S(d), A)\big)$ or, equivalently, a functor from $C(c, A) \times C\big(d, S_n(c)\big)$ to $C\big(S(d), A\big)$ which can be inductively defined by

$$\begin{tikzcd}
C(c, A) \times C\big(d, S_n(c)\big) \arrow[d, "\nu_n \times \mathrm{id}"] \\
C\big(S_n(c), A\big) \times C\big(d, S_n(c)\big) \arrow[d, "\mathrm{comp}"] \\
C(d, A) \arrow[d, "\nu_d"] \\
C\big(S(d), A\big)
\end{tikzcd}$$

Definition A.5. *Given algebraic structure (S, E), an (S, E)-*algebra *is an S-algebra that satisfies the equations, i.e., an S-algebra (A, ν) such that both legs of*

$$\begin{tikzcd}
C(c, A) \arrow[r, "\nu_\omega"] & C\big(S_\omega(Kc), A\big) \arrow[r, shift left, "C(\tau_{1c}, A)"] \arrow[r, shift right, "C(\tau_{2c}, A)"'] & C\big(E(c), A\big)
\end{tikzcd}$$

agree.

Given (S, E)–algebras (A, ν) and (B, δ), we define the hom-category

$$(S, E)\text{-}Alg\big((A, \nu), (B, \delta)\big)$$

to be the equaliser in C of

$$\begin{tikzcd}
C(A, B) \arrow[r, "\{C(S(c), -)\}_{c \in \operatorname{ob} C_f}"] \arrow[d, "\{C(S(c), -)\}_{c \in \operatorname{ob} C_f}"] & \prod_c C(C(c, A), C(c, B)) \arrow[d, "\prod_c C(C(c, A), \delta_c)"'] \\
\prod_c C(C(S(c), A), C(S(c), B)) \arrow[r, "\prod_c C(\nu_c, C(S(c), B))"'] & \prod_c C(C(c, A), C(S(c), B))
\end{tikzcd}$$

This agrees with our usual universal-algebraic understanding of the notion of homomorphism of algebras, internalising it to C. (S, E)-*Alg* can then be made into a 2-category in which composition is induced by that in C. An arrow in (S, E)-*Alg* is a functor $F\colon A \to B$ such that, for all finitely presentable c,

$$F\nu_c(-) = \delta_c(F-)\colon C(c, A) \longrightarrow C\big(S(c), B\big)$$

i.e., a functor that commutes with all basic c-ary operations for all c.

A special case of the main result of [KP93] says

Theorem A.6. *A 2-category is equivalent to (S, E)-Alg for algebraic structure (S, E) on C if and only if there is a finitary 2-monad T on C such that the 2-category is equivalent to T-Alg.*

See [Po97] for an account directed towards a computer science readership.

Reasoning about Idealized Algol Using Regular Languages

Dan R. Ghica[1,⋆] and Guy McCusker[2]

[1] Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, K7L 3N6; Fax: +1 (613) 533-6513; ghica@cs.queensu.ca
[2] School of Cognitive and Computing Sciences, University of Sussex at Brighton, Falmer, Brighton, UK, BN1 9QH; Fax: +44 (1273) 671320; guym@cogs.susx.ac.uk

Abstract. We explain how recent developments in game semantics can be applied to reasoning about equivalence of terms in a non-trivial fragment of Idealized Algol (IA) by expressing sets of complete plays as regular languages. Being derived directly from the fully abstract game semantics for IA, our method of reasoning inherits its desirable theoretical properties. The method is mathematically elementary and formal, which makes it uniquely suitable for automation. We show that reasoning can be carried out using only a meta-language of extended regular expressions, a language for which equivalence is formally decidable.
Keywords: Game semantics, Algol-like languages, regular languages

1 Introduction

Reynolds's Idealized Algol (IA) is a compact language which combines the fundamental features of procedural languages with a full higher-order procedure mechanism. This combination makes the language very expressive. For example, simple forms of classes and objects may be encoded in IA [14]. For these reasons, IA has attracted a great deal of attention from theoreticians; some 20 papers spanning almost 20 years of research were recently collected in book form [10].

A common theme in the literature on semantics of IA, beginning with [5], is the use of putative program equivalences to test suitability of semantic models. These example equivalences are intended to capture intuitively valid principles such as the privacy of local variables, irreversibility of state-changes and representation independence. A good model should support these intuitions.

Over the years, a variety of models have been proposed, each of which went some way towards formalizing programming intuition: functor categories gave an account of variable allocation and deallocation [11], relational parametricity was employed to capture representation-independence properties [9], and linear logic to explain irreversibility [8]. Recently, many of these ideas have been successfully incorporated in an operationally-based account of IA by Pitts [12].

⋆ This author acknowledges the support of a PGSB grant from the Natural Sciences and Engineering Research Council of Canada. This paper was written while visiting University of Edinburgh, Laboratory for Foundations of Computer Science.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 103–115, 2000.
© Springer-Verlag Berlin Heidelberg 2000

A frustrating situation was created with the development of a fully abstract game semantics for IA [1]. The full abstraction result means that the model validates all correct equivalences between programs, but unfortunately the model as originally presented is complicated, and calculating and reasoning within the model is difficult.

In this paper, we show that if one restricts attention to the second-order subset of IA, the games model can be simplified dramatically: terms now denote regular languages, and a relatively straightforward notation can be used to describe and calculate with the simplified semantics. The fragment of IA which we consider contains almost all the example equivalences from the literature, and we are therefore able to validate them in a largely calculational, algebraic style, using our semantics. We also obtain a decidability result for equivalence of programs in this fragment.

The approach of game semantics, and therefore of this paper, has little in common with the traditional semantics of IA. Intuitively it comes closest to Reddy's "object semantics" [13] and Brookes's trace semantics for shared-variable concurrent Algol [2]. Identifiers are not interpreted using an environment, variables are not interpreted using a notion of store and functions in the language are not interpreted using a mathematical notion of function. Instead, we are primarily concerned with behaviour, with all the possible actions that can be associated with every such language entity. Meanings of phrases are then constructed combinatorially according to the semantic rules of the language.

We believe our new presentation of game semantics is elementary enough to be considered a potential "popular semantics" [16]; it should at least provide a point of entry to game semantics for those who have previously found the subject opaque. Moreover, the property of full abstraction together with the fact that reasoning can be carried out in a decidable formal language suggest that our approach constitutes a good foundation on which an automatic program checker for IA and related languages can be constructed. The idea of using game semantics to support automated program analysis has already been independently explored in a more general framework by Hankin and Malacaria [3,4]. They used such models to derive static analysis algorithms which can be described without reference to games.

2 The IA Fragment

The principles of the programming language IA were laid down by John Reynolds in an influential paper [15]. IA is a language that combines imperative features with a procedure mechanism based on a typed call-by-name lambda calculus; local variables obey a stack discipline, having a lifetime dictated by syntactic scope; expressions, including procedures returning a value, cannot have side effects, *i.e.* they cannot assign to variables. We conform to these principles, except for the last one. This flavour of IA is known as IA with *active expressions* and has been analyzed extensively [18,1,8]. We consider only the recursion-free second order fragment of this language, the fragment which has been used to give

virtually all the significant equivalences mentioned in the literature. In addition, we will only deal with finite data sets.

The data types of the language (*i.e.* types of data assignable to variables) are a finite subset of the integers, and booleans:

$$\tau ::= \mathbf{int} \mid \mathbf{bool}$$

The phrase types of the language are those of commands, variables and expressions, plus function types.

$$\sigma ::= \mathbf{comm} \mid \mathbf{var}[\tau] \mid \mathbf{exp}[\tau] \ , \qquad \theta ::= \sigma \mid \sigma \to \theta$$

Note that we include only first-order function types here. We will consider only terms of the form

$$\iota_1 : \theta_1, \ldots, \iota_k : \theta_k \vdash M : \sigma$$

that is, terms of ground type with free variables of arbitrary first-order type. For the sake of simplicity in this paper, we also assume that M is β-normal, so that it contains no λ-abstractions. Function application is restricted to free identifiers ι. This last restriction can easily be removed, but at the expense of undue notational overhead in the semantics.

The terms of the language are as follows. In type **comm** there are basic commands **skip**, to do nothing, and Ω to diverge; in type **exp[int]** the finitary fragment contains constants n belonging to a finite subset $\mathcal{N}$ of the set of integers; and in type **exp[bool]** there are the constants **true** and **false**. There are term formers for assignment to variables, $V := E$, dereferencing variables, $!V$, sequential composition of commands $C; C'$, and sequential composition of a command with an expression to yield a possibly side-effecting expression $C; E$. We have a conditional operation **if** B **then** C **else** C', a while-loop **while** B **do** C, application of first-order identifiers to arguments $\iota M_1 \ldots M_k$, and the local-variable declaration **new**$[\tau]$ ι **in** C. Here, the free variable $\iota : \mathbf{var}[\tau]$ of C becomes bound. Finally, we assume the usual range of binary operations on integer and boolean expressions.

3 Game Semantics of Idealized ALGOL

In game semantics, a computation is represented as an *interaction* between two protagonists: *Player* (P) represents the program, and *Opponent* (O) represents the environment or context in which the program runs. For example, for a program of the form

$$\iota : \mathbf{exp}[\mathbf{int}] \to \mathbf{comm} \vdash M : \mathbf{comm} \ ,$$

Player will represent the program M; *Opponent* represents the context, in this case the non-local procedure ι. This procedure, if called by M, may in turn call an argument, in which case O will ask P to provide this information.

The interaction between O and P consists of a sequence of moves, alternating between players. In the game for the type **comm**, for example, there is an initial

move *run* to initiate a command, and a single response *done* to signal termination. Thus a simple interaction corresponding to the command **skip** might be

O: *run* (start executing)
P: *done* (immediately terminate).

In more interesting games, such as the one used to interpret programs like

$$\iota : \mathbf{exp}[\mathbf{int}] \to \mathbf{comm} \vdash \iota(0) : \mathbf{comm} \ ,$$

there are more moves. Corresponding to the result type **comm**, there are the moves *run* and *done*. The program needs to run the procedure ι, so there are also moves run_ι and $done_\iota$ to represent that; here the run_ι move is a move for P, and $done_\iota$ is a move for O. Finally, the procedure ι may need to evaluate its argument. For this purpose, O has a move q_ι^1, meaning "what is the value of the first argument to ι?", to which P may respond with an integer n, tagged as n_ι^1 for the sake of identification.

Here is a sample interaction in the interpretation of the above term.

O: *run* (start executing)
P: run_ι (execute ι)
O: q_ι^1 (what is the first argument to ι?)
P: 0_ι^1 (the argument is 0)
O: $done_\iota$ (ι terminates)
P: *done* (whole command terminates).

In the above interaction, at the third move, O was not compelled to ask for the argument to ι: if O represented a non-strict procedure, the move $done_\iota$ would be played immediately. Similarly, at the fifth move, O could repeat the question q_ι to represent a procedure which calls its argument more than once.

Strategies. Using the above ideas, each possible execution of a program is represented as a sequence of moves in the appropriate game. A *program* can therefore be represented as a *strategy* for P, that is, a predetermined way of responding to the moves O makes. A strategy can also choose to make no response in a particular situation, representing divergence, so for example there are two strategies for the game corresponding to **comm**: the strategy for **skip** responds to *run* with *done*, and the strategy for Ω fails to respond to *run* at all.

Strategies are usually represented as *sets of sequences of moves*, so that a strategy is identified with the collection of possible traces that can arise if P plays according to that strategy. The fact that O can repeat questions, as we remarked above, means that these sets are very often infinite, even for simple programs. The strategy for the program $\iota(0)$, for example, is capable of supplying the argument 0 to ι as often as O asks for it.

Interpretation of Variables. The type $\mathbf{var}[\tau]$ is represented as a game in the following way. For each element x of τ there is an initial move $write(x)$, representing an assignment. There is one possible response to this move, *ok*, which signals successful completion of the assignment. For dereferencing, there is an initial move *read*, to which P may respond with any element of τ.

Here is an interaction in the strategy for

$$v : \mathbf{var}[\mathbf{int}] \vdash v := !v + 1.$$

O: *run*
P: $read_v$ (get the value from v)
O: 3 (O supplies the value 3)
P: $write(4)_v$ (write 4 into v)
O: ok_v (the assignment is complete)
P: *done* (the whole command is complete)

In these interactions, O is *not* constrained to play a *good variable* in v, *i.e.* to exhibit the expected causal dependency between reads and writes. For example, in the game for terms of the form

$$c : \mathbf{comm}, v : \mathbf{var}[\mathbf{int}] \vdash M : \mathbf{comm} \ ,$$

we find interactions such as

$$run \cdot read_v \cdot 3_v \cdot write(4)_v \cdot ok_v \cdot run_c \cdot done_c \cdot read_v \cdot 7_v \cdots$$

Here O has not played a good variable in v, but this freedom is necessary. Our semantics must take care of the case in which ι is bound to a procedure which also uses v, for example, the procedure $v := 7$.

There is one situation in which this kind of interference cannot happen: when the variable v is made local. This has two effects. The local interaction with v is guaranteed to exhibit "good variable" behaviour, and the interaction with v is not an observable part of the programs behaviour. Therefore, the games interpretation of **new** v **in** M is given by taking the set of sequences interpreting M, considering only those in which O plays a good variable in v, and deleting all the moves pertaining to v, to hide v from the outside.

Full abstraction. In [1], it was shown that games give rise to a fully abstract model of IA, in the following sense. Say that an interaction is *complete* if and only if it begins with an initial move and ends with a move which answers that initial move. Thus, for example, $run{\cdot}run_\iota$ is not complete but $run{\cdot}run_\iota{\cdot}done_\iota{\cdot}done$ is. Then we have the following theorem:

Theorem 1 (Full Abstraction for IA). *For any $\Gamma \vdash P, Q : \theta$, programs P and Q are contextually equivalent in IA ($P \equiv Q$) if and only if the sets of complete plays in the strategies interpreting P and Q are equal.*

Note. In the above account, a very simple notion of game has been used. In fact, games models require a great deal more machinery, including the notions of *justification pointer* and *questions and answers*, in order for full abstraction to be achieved. The key observation which makes the present paper possible is that, for the interpretation of IA up to second-order types, this extra machinery is redundant; it only comes into play at third-order and above.

4 Regular Language Game Semantics

We will now give a simple presentation of the game semantics of our fragment of IA. The key idea is that the set of complete plays in a strategy forms a regular language, which leads to a compact notation for defining and manipulating these infinite sets of sequences. We define a metalanguage based on regular expressions, extended with two handy operations: intersection and hiding. Of course, these extensions do not change the regular nature of the languages being defined.

Definition 1. *The set $\mathcal{R}_{\mathcal{A}}$ of extended regular expressions over a finite alphabet $\mathcal{A}$ is defined inductively as the smallest set for which:*

Constants: $\bot, \epsilon \in \mathcal{R}_{\mathcal{A}}$; *if* $a \in \mathcal{A}$, *then* $a \in \mathcal{R}_{\mathcal{A}}$;
Iteration: *if* $R \in \mathcal{R}_{\mathcal{A}}$, $R^* \in \mathcal{R}_{\mathcal{A}}$;
Operators: *if* $R, S \in \mathcal{R}_{\mathcal{A}}$, *then* $R{\cdot}S, R + S, R \cap S \in \mathcal{R}_{\mathcal{A}}$;
Hiding: *if* $R \in \mathcal{R}_{\mathcal{A}}$, $\mathcal{A}' \subseteq \mathcal{A}$, *then* $R\,|_{\mathcal{A}'} \in \mathcal{R}_{\mathcal{A}}$;

The constant $\bot$ denotes the empty language, while ϵ is the language consisting only of the empty string. The constant a is the language consisting of the singleton sequence a. Hiding represents the operation of restricting a language to a subset $\mathcal{A} \setminus \mathcal{A}'$ of the original alphabet $\mathcal{A}$: the language $\mathcal{L}(R\,|_{\mathcal{A}'})$ is the set of sequences in $\mathcal{L}(R)$, with all elements of $\mathcal{A}'$ deleted. The other operations (iteration, concatenation, union, intersection) are defined as usual.

Proposition 1. *Every extended regular expression denotes a regular language.*

We now give a regular language representation of the game semantics for IA. An alphabet is associated with every type in IA. They represent a semantic "domain" over which regular languages will be constructed, using extended regular expressions:

$$\begin{aligned}
&\mathcal{A}[\![\mathbf{int}]\!] = \mathcal{N}, \quad \mathcal{A}[\![\mathbf{bool}]\!] = \{\mathit{true}, \mathit{false}\}\\
&\mathcal{A}[\![\mathbf{comm}]\!] = \{\mathit{run}, \mathit{done}\},\\
&\mathcal{A}[\![\mathbf{exp}[\tau]]\!] = \{q, v \mid v \in \mathcal{A}[\![\tau]\!]\},\\
&\mathcal{A}[\![\mathbf{var}[\tau]]\!] = \{\mathit{read}, v, \mathit{write}(v), \mathit{ok} \mid v \in \mathcal{A}[\![\tau]\!]\},\\
&\mathcal{A}[\![\sigma_1 \to \sigma_2 \to \ldots \to \sigma_k \to \sigma]\!] = \{a^i \mid a \in \mathcal{A}[\![\sigma_i]\!], 1 \le i \le k\} \cup \mathcal{A}[\![\sigma]\!].
\end{aligned}$$

By a^k we mean a lexical operation: the creation of a new symbol by tagging the symbol a with the numeral k.

For a term of the form

$$\iota_1 : \theta_1, \iota_2 : \theta_2, \ldots \iota_k : \theta_k \vdash M : \sigma$$

we define the *context alphabet* to be the set

$$\bigcup_{1 \le j \le k} \{a_{\iota_j} \mid a \in \mathcal{A}[\![\theta_j]\!]\}$$

that is, the union of the $\mathcal{A}[\![\theta_j]\!]$ alphabets, every symbol tagged with the corresponding identifier.

The semantics of a term M as above is then a regular language of a certain form, defined as follows.

- If $\sigma = \mathbf{comm}$, $[\![M]\!] = run \cdot R_M \cdot done$.
- If $\sigma = \mathbf{exp}[\tau]$, $[\![M]\!] = \sum_{v \in \mathcal{A}[\![\tau]\!]} q \cdot R_M^v \cdot v$
- If $\sigma = \mathbf{var}[\tau]$,

$$[\![M]\!] = \sum_{v \in \mathcal{A}[\![\tau]\!]} (read \cdot R_M^v \cdot v) + \sum_{v \in \mathcal{A}[\![\tau]\!]} (write(v) \cdot S_M^v \cdot ok)$$

where R_M, R_M^v and S_M^v are regular languages over the context alphabet of the term M. The idea is that, for M of type **comm**, for example, the regular language R_M is the set of interactions with the environment that need to take place for M to terminate. Similarly, R_M^3 is the set of interactions that an expression M must have with the environment to return a value of 3, and so on. For M of type $\mathbf{var}[\tau]$, R_M^v denotes the interactions required for a value v to be read from M, and S_M^v denotes the interactions needed to write v into M.

These regular languages, denoted by R_M, R_M^v, S_M^v, form the substance of our interpretation of the language; the moves that bracket them, such as *run*, *done* for commands, are merely delimiters to indicate that a complete play has occurred. The definitions needed to interpret most of our language are given in Table 1.

Table 1. Some semantic valuations

$$R_{\mathbf{skip}} = \epsilon \qquad R_\Omega = \bot \qquad R_v^v = \epsilon \qquad R_v^{v'} = \bot \quad (v \neq v')$$

$$R_{\iota:\mathbf{comm}} = run_\iota \cdot done_\iota \qquad R_{\iota:\mathbf{exp}[\tau]}^v = q_\iota \cdot v_\iota$$

$$R_{\iota:\mathbf{var}[\tau]}^v = read_\iota \cdot v_\iota \qquad S_{\iota:\mathbf{var}[\tau]}^v = write(v)_\iota \cdot ok_\iota$$

$$R_{\mathbf{while}\ B\ \mathbf{do}\ C} = (R_B^{true} \cdot R_M)^* \cdot R_B^{false} \qquad R_{E_1+E_2}^n = \sum_{n_1+n_2=n} R_{E_1}^{n_1} \cdot R_{E_2}^{n_2}$$

$$R_{E_1=E_2}^{true} = \sum_{n \in \mathcal{N}} R_{E_1}^n \cdot R_{E_2}^n \qquad R_{E_1=E_2}^{false} = \sum_{n1 \neq n2} R_{E_1}^{n_1} \cdot R_{E_2}^{n_2}$$

$$R_{\mathbf{if}\ B\ \mathbf{then}\ C\ \mathbf{else}\ C'} = R_B^{true} \cdot R_C + R_B^{false} \cdot R_{C'} \qquad R_{C;C'} = R_C \cdot R_{C'}$$

$$R_{!V}^v = R_V^v \qquad R_{V:=M} = \sum_v R_M^v \cdot S_V^v$$

For instance, a trace of $V := E$ consists of *run* and *done* surrounding the effects of the assignment: first R_E^v which is the regular language denoting the

interaction which leads the expression E to return value v, and then S_V^v which is the regular language denoting the interaction required to write value v into variable V.

A trace of a while-loop has the form: some number of repetitions of a trace of the guard which produces *true* followed by a complete trace of the loop body, then, finally, a single trace of the guard producing *false*. Using our semantics, we can easily demonstrate the validity of a typical while-loop equivalence:

$$\begin{aligned}
&[\![\textbf{while true do } C]\!] \\
&\quad = run \cdot \left(R^{true}_{\textbf{true}} \cdot R_C\right)^* \cdot R^{false}_{\textbf{true}} \cdot done \\
&\quad = run \cdot (\epsilon \cdot R_C)^* \cdot \bot \cdot done \\
&\quad = \bot = [\![\Omega]\!].
\end{aligned}$$

The semantics of a free identifier ι consist simply of querying the identifier. There is no need to look up the identifier in an environment, because the tagging of the trace with the name of the identifier ensures the proper correspondence between each identifier and its effects. Therefore, a notion of environment is not needed here at all.

The semantics of application and of local variables have been omitted from Table 1 because they deserve additional explanation.

Application. Let ι be a free variable of type $\sigma_1 \to \sigma_2 \to \ldots \to \sigma_k \to \textbf{comm}$, and $M_1, \ldots, M_k$ be terms of type $\sigma_1, \ldots, \sigma_k$. The interpretation of the application $\iota M_1 \ldots M_k$ depends on the moves available, which depends on the types $\sigma_1, \ldots, \sigma_k$. In the simplest case, when every σ_j is the type **comm**, we define

$$R_{\iota M_1 \ldots M_k} = run_\iota \cdot \left(\sum_{j=1}^{k} run_\iota^j \cdot R_{M_j} \cdot done_\iota^j \right)^* \cdot done_\iota.$$

To illustrate a more complex case, we give the definition of the interpretation of ιM where ι has type $\textbf{var}[\textbf{int}] \to \textbf{exp}[\textbf{int}]$.

$$R^v_{\iota M} = q_\iota \cdot \left(\sum_n read_\iota^1 \cdot R_M^n \cdot n_\iota^1 + \sum_n write(n)_\iota^1 \cdot S_M^n \cdot ok_\iota^1 \right)^* \cdot v_\iota.$$

The large sums in this expression show that the environment chooses how to read and write from the argument to ι, and that the term M determines what behaviour results from such reading and writing.

In general, for a variable $\iota : \sigma_1 \to \sigma_2 \to \ldots \sigma_k \to \textbf{comm}$:

$$R_{\iota M_1 \ldots M_k} = run_\iota \cdot \left(\sum_{j=1}^{k} \rho_\iota^j [\![M_j]\!] \right)^* \cdot done_\iota$$

where ρ_ι^j is a relabeling operation that tags the initial and final moves of the arguments M_j, the bracketing indicating a complete play, with the identifier which is calling them and the position in which they are used:

$$\rho_\iota^j(R) = R[w_\iota^j / w], \text{ for } w \in \{run, done, q, v, read, write(v), ok \mid v \in \mathcal{A}[\![\tau]\!]\}.$$

Local variables. For the semantics of a local variable block, as in the original game semantics, there are two things to do: restrict O's behaviour to that of a good variable, and hide the interaction with the local variable.

The regular language γ_ι^τ stipulates that the moves corresponding to ι have good-variable behaviour. First, let $\mathcal{A}[\![\tau]\!]_\iota$ be that part of the alphabet which concerns the variable $\iota : \mathbf{var}[\tau]$, that is,

$$\mathcal{A}[\![\tau]\!]_\iota = \{read_\iota, v_\iota, write(v)_\iota, ok_\iota \mid v \in \mathcal{A}[\![\tau]\!]\}.$$

Let $B_\iota = (\sum_{x \notin \mathcal{A}[\![\tau]\!]_\iota} x)^*$ be the regular language containing all strings which do not contain any elements of $\mathcal{A}[\![\tau]\!]_\iota$. If we assume that variables initially hold some default value a^τ, then good-variable behaviour is stipulated as follows.

$$\gamma_\tau^\iota = B_\iota \cdot (read_\iota \cdot a_\iota^\tau \cdot B_\iota)^* \cdot \left(B_\iota \cdot \sum_{v \in \mathcal{A}[\![\tau]\!]} \left(write(v)_\iota \cdot ok \cdot B_\iota \cdot (read_\iota \cdot v_\iota \cdot B_\iota)^* \right) \right)^*$$

For the sake of completeness, $a^{\mathbf{int}} = 0$ and $a^{\mathbf{bool}} = \mathit{false}$. We can then give the semantics of blocks as

$$R_{\mathbf{new}[\tau]\ \iota\ \mathbf{in}\ M} = (\gamma_\iota^\tau \cap R_M) \mid_{\mathcal{A}[\![\tau]\!]_\iota} .$$

Note that the same intersection and hiding can be used to define $[\![\mathbf{new}[\tau]\ \iota\ \mathbf{in}\ M]\!]$ directly from $[\![M]\!]$: the bracketing moves, *run* and *done*, make no difference.

$$[\![\mathbf{new}[\tau]\ \iota\ \mathbf{in}\ M]\!] = (\gamma_\tau^\iota \cap [\![M]\!]) \mid_{\mathcal{A}[\![\tau]\!]_\iota} .$$

Theorem 2. Full abstraction. *Two terms of the recursion free second order finitary fragment of IA are equivalent (in full IA) if and only if the languages denoted by them are equal:*

For any $\Gamma \vdash P, Q : \theta$, $\quad P \equiv Q \iff [\![P]\!] = [\![Q]\!]$.

Proof. We can show that the regular language denoted by a term of IA is equal to the set of complete plays in the fully abstract game semantics [1], therefore the full abstraction property is preserved. Note that language equivalence is asserted outside the fragment we describe here; witnesses to some inequivalences may belong to IA but not to the presented fragment. □

5 Examples of Reasoning

At this point a skeptical reader may entertain doubts concerning our earlier claim of simplicity. We have set up a formal notation of extended regular expressions which includes rather complicated operations. However, the complications are notational and not conceptual. Also, all the operations involved are defined effectively so carrying them out is a mechanical process. We hope that the simplicity of our approach will become clearer when we show examples of reasoning about putative equivalences.

Locality. This most simple of equivalences invalidates models of imperative computation relying on a global store, traceable back to Scott and Strachey [17]. It says that a globally defined procedure cannot modify a local variable, and it was first proved using the "possible worlds" model of Reynolds and Oles, constructed using functor categories [11].

$$P : \mathbf{comm} \vdash \ \mathbf{new}\ x\ \mathbf{in}\ P \equiv P$$

Proof.

$$\begin{aligned}
[\![\mathbf{new}\ x\ \mathbf{in}\ P]\!] &= (\gamma^x \cap [\![P]\!])\mid_{\mathcal{A}_x} \\
&= (\gamma^x \cap \mathit{run}\cdot \mathit{run}_P \cdot \mathit{done}_P \cdot \mathit{done})\mid_{\mathcal{A}_x} \\
&= (\mathit{run}\cdot \mathit{run}_P \cdot \mathit{done}_P \cdot \mathit{done})\mid_{\mathcal{A}_x} \\
&\quad \text{because no moves are tagged by } x \\
&= \mathit{run}\cdot \mathit{run}_P \cdot \mathit{done}_P \cdot \mathit{done} \\
&= [\![P]\!]
\end{aligned}$$

Snapback. This example captures the intuition that changes to the state are in some way irreversible. A procedure executing an argument which is a command inflicts upon the state changes that cannot be undone from within the procedure. This is why, in the following, if procedure P uses its argument both sides will fail to terminate; if procedure P does not use its argument the behaviour of each side will be identical because of the locality of x, as seen above. The first model to address this is sue correctly was O'Hearn and Reynolds's interpretation of IA using the polymorphic linear lambda calculus [8]. Reddy also addressed this issue using a novel "object semantics" approach [13], but in a particular flavour of IA known as interference-controlled ALGOL [6]. A further development of this model, that also satisfies this equivalence, is O'Hearn and Reddy's [7], a model fully abstract for the second order subset.

$$\begin{aligned}
&P : \mathbf{comm} \to \mathbf{comm} \vdash \\
&\ \mathbf{new}\ x\ \mathbf{in}\ P(x := 1);\ \mathbf{if}\ !x = 1\ \mathbf{then}\ \Omega\ \mathbf{else\ skip} \equiv P(\Omega)
\end{aligned}$$

Proof.

$$[\![x := 1]\!] = \mathit{run}\cdot \mathit{write}(1)_x \cdot \mathit{ok}_x \cdot \mathit{done}$$

$$[\![P(x := 1)]\!] = \mathit{run}\cdot \mathit{run}_P \cdot \left(\mathit{run}^1_P \cdot \mathit{write}(1)_x \cdot \mathit{ok}_x \cdot \mathit{done}^1_P\right)^* \cdot \mathit{done}_P \cdot \mathit{done}$$

$$[\![\mathbf{if}\ !x = 1\ \mathbf{then}\ \Omega\ \mathbf{else\ skip}]\!] = \sum_{n\neq 1} \mathit{run}\cdot \mathit{read}_x \cdot n_x \cdot \mathit{done}$$

$$\begin{aligned}
&[\![P(x := 1);\ \mathbf{if}\ !x = 1\ \mathbf{then}\ \Omega\ \mathbf{else\ skip}]\!] \\
&= \mathit{run}\cdot \mathit{run}_P \cdot \left(\mathit{run}^1_P \cdot \mathit{write}(1)_x \cdot \mathit{ok}_x \cdot \mathit{done}^1_P\right)^* \cdot \mathit{done}_P \cdot \left(\sum_{n\neq 1} \mathit{read}_x \cdot n_x\right) \cdot \mathit{done}
\end{aligned}$$

$$\begin{aligned}
&\gamma^x \cap [\![P(x := 1);\ \mathbf{if}\ !x = 1\ \mathbf{then}\ \Omega\ \mathbf{else\ skip}]\!] \\
&\quad = \mathit{run}\cdot \mathit{run}_P \cdot \mathit{done}_P \cdot \mathit{read}_x \cdot 0_x \cdot \mathit{done},
\end{aligned}$$

because the only possibility to complete a trace in $\sum_{n\neq 1} read_x{\cdot}n_x$ is if the trace in $\left(run^1_P{\cdot}write(1)_x{\cdot}ok_x{\cdot}done^1_P\right)^*$ is the empty trace. Otherwise, the good variable property of x requires $n_x = 1_x$, which is banned by the set to which n is restricted ($n \neq 1$). The meaning of the left hand term of the equivalence is therefore:

$$\begin{aligned}&(\gamma^x \cap [\![P(x := 1);\ \textbf{if } !x = 1 \textbf{ then } \Omega \textbf{ else skip}]\!]) \mid_{\mathcal{A}_x}\\ &\quad = run{\cdot}run_P{\cdot}done_P{\cdot}done = [\![P(\Omega)]\!]\end{aligned}$$

Parametricity. The intuition of parametricity is one of representation independence. Procedures passed different but equivalent implementations of a data structure or algorithm are not supposed to be able to distinguish between them. Several such motivating examples are given by O'Hearn and Tennent [9], who introduce a model constructed using a certain relation-preserving functor category.

The specific example we give is of the equivalence of two implementations of a toggle-switch: one which uses 1 for "on" and -1 for "off", and one which uses **true** and **false**. The semantic equations for negation and the inequality test have not been spelled out but are the obvious ones.

$$\begin{aligned}&P : \textbf{comm} \to \textbf{exp}[\textbf{bool}] \to \textbf{comm} \vdash\\ &\textbf{new}[\textbf{int}]\ x\ \textbf{in}\ x := 1; P(x := -!x)(!x > 0)\\ &\qquad \equiv \textbf{new}[\textbf{bool}]\ x\ \textbf{in}\ x := \textbf{true}\ ; P(x := \textbf{not}\ x)(!x)\end{aligned}$$

Proof.

$$\begin{aligned}&[\![x := -!x]\!] = \sum_{n\in\mathcal{N}} run{\cdot}read_x{\cdot}n_x{\cdot}write(-n)_x{\cdot}ok_x{\cdot}done\\ &[\![!x > 0]\!] = \sum_{n>0} q{\cdot}read_x{\cdot}n_x{\cdot}true + \sum_{n\leq 0} q{\cdot}read_x{\cdot}n_x{\cdot}false\\ &[\![x := 1; P(x := -!x)(!x > 0)]\!] = run{\cdot}write(1)_x{\cdot}ok_x{\cdot}\\ &\qquad run_P{\cdot}\left(\sum_{n\in\mathcal{N}} run^1_P{\cdot}read_x{\cdot}n_x{\cdot}write(-n)_x{\cdot}ok_x{\cdot}done^1_P + \right.\\ &\qquad \left.\sum_{n>0} q^2_P{\cdot}read_x{\cdot}n_x{\cdot}true^2_P + \sum_{n\leq 0} q^2_P{\cdot}read_x{\cdot}n_x{\cdot}false^2_P\right)^*{\cdot}done_P{\cdot}done\\ &\gamma^x_{\textbf{int}} \cap [\![x := 1; P(x := -!x)(!x > 0)]\!] =\\ &\quad = run{\cdot}write(1)_x{\cdot}ok_x{\cdot}run_P{\cdot}(\epsilon + X + X{\cdot}Y + X{\cdot}Y{\cdot}X + \ldots){\cdot}done_P{\cdot}done\\ &\quad = run{\cdot}write(1)_x{\cdot}ok_x{\cdot}run_P{\cdot}\left(X + (X{\cdot}Y)^*{\cdot}(X + \epsilon)\right){\cdot}done_P{\cdot}done\\ &\quad \text{where } X{=}run^1_P{\cdot}read_x{\cdot}1_x{\cdot}write(-1)_x{\cdot}ok_x{\cdot}done^1_P{\cdot}\left(q^2_P{\cdot}read_x{\cdot}(-1)_x{\cdot}false^2_P\right)^*\\ &\quad \text{and } Y{=}run^1_P{\cdot}read_x{\cdot}(-1)_x{\cdot}write(1)_x{\cdot}ok_x{\cdot}done^1_P{\cdot}\left(q^2_P{\cdot}read_x{\cdot}(1)_x{\cdot}true^2_P\right)^*\end{aligned}$$

Why this is the case should be intuitively clear. A value of 1 is written into x, followed by negation only, which constrains all the plays to $(+1)_x$ and $(-1)_x$

only. The reads and writes have to match with the good variable behaviour. A fully formal proof is lengthier but trivial and mechanical. Restricting with $|_{\mathcal{A}[\![\mathbf{int}]\!]_x}$ gives the following trace for the left hand side:

$$run \cdot run_P \cdot \big(X' + (X' \cdot Y')^* \cdot (X' + \epsilon)\big) \cdot done_P \cdot done$$
$$\text{where } X' = run^1_P \cdot done^1_P \cdot \big(q^2_P \cdot false^2_P\big)^* \text{ and } Y' = run^1_P \cdot done^1_P \cdot \big(q^2_P \cdot true^2_P\big)^*$$

A similar calculation on the right hand side leads to the the same result.

6 Decidability and Complexity Issues

As we have seen, regular languages provide a semantics for the fragment of IA described here. To manipulate regular languages we have introduced a formal meta-language of extended regular expressions, which preserves regularity of the language. All the operations we have used in formulating the semantic valuations have been effectively given. Therefore, we can formulate the following obvious result:

Theorem 3 (Decidability). *Equivalence of two terms of the recursion free second order finitary fragment of IA is decidable.*

For the general problem of term equivalence the complexity bound appears to be at least of exponential space, as is the case for regular expressions with intersection [19]. However, the complexity bound for the general problem may not be relevant for the kind of terms that arise in the model of IA, and particularly for those that would be checked for equivalence in practice. This point, which will be investigated in future work, is of the utmost importance if a tool is to be developed based on our ideas.

Acknowledgments We are grateful to Robert Tennent, Samson Abramsky and Mark Jerrum for suggestions, questions and advice. We thank Peter O'Hearn and Pasquale Malacaria for a stimulating discussion on the paper, and our anonymous referees for their insightful comments.

References

1. S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions (extended abstract). In *Proceedings of 1996 Workshop on Linear Logic*, volume 3 of *Electronic notes in Theoretical Computer Science*. Elsevier, 1996. Also as Chapter 20 of [10].
2. S. Brookes. Full abstraction for a shared variable parallel language. In *Proceedings, 8th Annual IEEE Symposium on Logic in Computer Science*, pages 98–109, Montreal, Canada, 1993. IEEE Computer Society Press, Los Alamitos, California.
3. C. Hankin and P. Malacaria. Generalised flowcharts and games. *Lecture Notes in Computer Science*, 1443, 1998.
4. C. Hankin and P. Malacaria. A new approach to control flow analysis. *Lecture Notes in Computer Science*, 1383, 1998.

5. A. R. Meyer and K. Sieber. Towards fully abstract semantics for local variables: preliminary report. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 191–203, San Diego, California, 1988. ACM, New York. Reprinted as Chapter 7 of [10].
6. P. W. O'Hearn, A. J. Power, M. Takeyama, and R. D. Tennent. Syntactic control of interference revisited. *Theoretical Computer Science*, 228:175–210, 1999. Preliminary version reprinted as Chapter 18 of [10].
7. P. W. O'Hearn and U. S. Reddy. Objects, interference and the Yoneda embedding. In S. Brookes, M. Main, A. Melton, and M. Mislove, editors, *Mathematical Foundations of Programming Semantics, Eleventh Annual Conference*, volume 1 of *Electronic Notes in Theoretical Computer Science*, Tulane University, New Orleans, Louisiana, Mar. 29–Apr. 1 1995. Elsevier Science (http://www.elsevier.nl).
8. P. W. O'Hearn and J. C. Reynolds. From Algol to polymorphic linear lambda-calculus. Journal of the Association for Computing Machinery, to appear.
9. P. W. O'Hearn and R. D. Tennent. Relational parametricity and local variables. In *Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 171–184, Charleston, South Carolina, 1993. ACM, New York. A version also published as Chapter 16 of [10].
10. P. W. O'Hearn and R. D. Tennent, editors. ALGOL-*like Languages*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1997. Two volumes.
11. F. J. Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages*. Ph.D. thesis, Syracuse University, Syracuse, N.Y., 1982.
12. A. M. Pitts. Reasoning about local variables with operationally-based logical relations. In *11th Annual Symposium on Logic in Computer Science*, pages 152–163. IEEE Computer Society Press, Washington, 1996. A version also published as Chapter 17 of [10].
13. U. S. Reddy. Global state considered unnecessary: Introduction to object-based semantics. LISP *and Symbolic Computation*, 9(1):7–76, 1996. Published also as Chapter 19 of [10].
14. J. C. Reynolds. Syntactic control of interference. In *Conference Record of the Fifth Annual ACM Symposium on Principles of Programming Languages*, pages 39–46, Tucson, Arizona, Jan. 1978. ACM, New York.
15. J. C. Reynolds. The essence of ALGOL. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages,* Proceedings of the International Symposium on Algorithmic Languages, pages 345–372, Amsterdam, Oct. 1981. North-Holland, Amsterdam. Reprinted as Chapter 3 of [10].
16. D. A. Schmidt. On the need for a popular formal semantics. *ACM SIGPLAN Notices*, 32(1):115–116, Jan. 1997.
17. D. S. Scott and C. Strachey. Toward a mathematical semantics for computer languages. In J. Fox, editor, *Proceedings of the Symposium on Computers and Automata*, volume 21 of *Microwave Research Institute Symposia Series*, pages 19–46. Polytechnic Institute of Brooklyn Press, New York, 1971. Also Technical Monograph PRG-6, Oxford University Computing Laboratory, Programming Research Group, Oxford.
18. K. Sieber. Full abstraction for the second order subset of an ALGOL-like language. In *Mathematical Foundations of Computer Science*, volume 841 of *Lecture Notes in Computer Science*, pages 608–617, Kŏsice, Slovakia, Aug. 1994. Springer-Verlag, Berlin. A version also published as Chapter 15 of [10].
19. L. J. Stockmeyer. The complexity of decision problems in automata theory and logic. Technical Report MIT/LCS/TR-133, Massachusetts Institute of Technology, Laboratory for Computer Science, July 1974.

The Measurement Process in Domain Theory

Keye Martin

Department of Mathematics, Tulane University, New Orleans, LA 70118, USA
Fax (504)865-5063
martin@math.tulane.edu
http://www.math.tulane.edu/~martin

Abstract. We introduce the measurement idea in domain theory and then apply it to establish two fixed point theorems. The first is an extension of the Scott fixed point theorem which applies to nonmonotonic mappings. The second is a contraction principle for monotone maps that guarantees the existence of *unique* fixed points.

1 Introduction

A measurement on a domain D is a Scott continuous map $\mu : D \to [0, \infty)^*$ into the nonnegative reals in their *reverse* order which formalizes the notion *information content* for objects in a domain. Intuitively, if $x \in D$ is an *informative* object, then μx is the *amount of information* it contains. In another light, we may think of μ as measuring the disorder in an object, or *entropy,* since $x \sqsubseteq y \Rightarrow \mu x \geq \mu y$, that is, the more informative an object is, the smaller its measure.

After giving a precise definition of measurement and several natural examples, we show the value of the idea by proving and then applying two fixed point theorems. The first is an extension of the Scott fixed point theorem which applies to nonmonotonic processes, like the bisection method and the r-section search. The second is a contraction principle for monotone maps that guarantees the existence of *unique* fixed points, as opposed to the *least* fixed points that domain theory usually provides.

2 Background

2.1 Domain Theory

A *poset* is a partially ordered set [1].

Definition 1. A *least element* in a poset $(P, \sqsubseteq)$ is an element $\perp \in P$ such that $\perp \sqsubseteq x$ for all $x \in P$. Such an element is unique. An element $x \in P$ is *maximal* if $(\forall y \in P)\ x \sqsubseteq y \Rightarrow x = y$. The set of maximal elements in a poset is written $\max P$.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 116–126, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Definition 2. Let $(P, \sqsubseteq)$ be a poset. A nonempty subset $S \subseteq P$ is *directed* if $(\forall x, y \in S)(\exists z \in S)\, x, y \sqsubseteq z$. The *supremum* of a subset $S \subseteq P$ is the least of all its upper bounds provided it exists. This is written $\bigsqcup S$. A *dcpo* is a poset in which every directed subset has a supremum.

Definition 3. In a poset $(P, \sqsubseteq)$, $a \ll x$ iff for all directed subsets $S \subseteq D$ which have a supremum, $x \sqsubseteq \bigsqcup S \Rightarrow (\exists s \in S)\, a \sqsubseteq s$. We set $\Downarrow x = \{a \in P : a \ll x\}$. An element $x \in P$ is *compact* if $x \ll x$. The set of compact elements in P is $K(P)$.

Definition 4. A subset B of a poset P is a *basis* for P if $B \cap \Downarrow x$ contains a directed subset with supremum x, for each $x \in P$.

Definition 5. A poset is *continuous* if it has a basis. A poset is *algebraic* if its compact elements form a basis. A poset is *ω-continuous* if it has a countable basis.

Definition 6. A *domain* is a continuous dcpo.

Definition 7. A subset U of a poset P is *Scott open* if

(i) U is an upper set: $x \in U$ & $x \sqsubseteq y \Rightarrow y \in U$, and
(ii) U is inaccessible by directed suprema: For every directed $S \subseteq P$ with a supremum,

$$\bigsqcup S \in U \Rightarrow S \cap U \neq \emptyset.$$

The collection of all Scott open subsets of P is called the Scott topology. It is denoted σ_P.

Unless explicitly stated otherwise, all topological statements about posets are made with respect to the Scott topology.

Proposition 1. *A function $f : D \to E$ between dcpos is continuous iff*

(i) *f is monotone: $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$.*
(ii) *f preserves directed suprema: For all directed $S \subseteq D$, $f(\bigsqcup S) = \bigsqcup f(S)$.*

2.2 Examples of Domains

Example 1. The *interval domain* is the collection of compact intervals of the real line

$$\mathbf{I}\mathbb{R} = \{[a, b] : a, b \in \mathbb{R} \ \&\ a \leq b\}$$

ordered under reverse inclusion

$$[a, b] \sqsubseteq [c, d] \Leftrightarrow [c, d] \subseteq [a, b]$$

is an ω-continuous dcpo. The supremum of a directed set $S \subseteq \mathbf{I}\mathbb{R}$ is $\bigcap S$, while the approximation relation is characterized by $I \ll J \Leftrightarrow J \subseteq \mathrm{int}(I)$. A countable basis for $\mathbf{I}\mathbb{R}$ is given by $\{[p, q] : p, q \in \mathbb{Q} \ \&\ p \leq q\}$.

Definition 8. A *partial function* $f : X \rightharpoonup Y$ between sets X and Y is a function $f : A \to Y$ defined on a subset $A \subseteq X$. We write $\mathrm{dom}(f) = A$ for the *domain* of a partial map $f : X \rightharpoonup Y$.

Example 2. The set of *partial mappings on the naturals*

$$[\mathbb{N} \rightharpoonup \mathbb{N}] = \{\, f \mid f : \mathbb{N} \rightharpoonup \mathbb{N} \,\}$$

becomes an ω-algebraic dcpo when ordered by extension

$$f \sqsubseteq g \Leftrightarrow \mathrm{dom}(f) \subseteq \mathrm{dom}(g) \ \&\ f = g \text{ on } \mathrm{dom}(f).$$

The supremum of a directed set $S \subseteq [\mathbb{N} \to \mathbb{N}]$ is $\bigcup S$, under the view that functions are certain subsets of $\mathbb{N} \times \mathbb{N}$, while the approximation relation is

$$f \ll g \Leftrightarrow f \sqsubseteq g \ \&\ \mathrm{dom}(f) \text{ is finite.}$$

The maximal elements of $[\mathbb{N} \to \mathbb{N}]$ are the *total functions,* that is, those functions f with $\mathrm{dom}(f) = \mathbb{N}$.

Example 3. The *Cantor set model* is the collection of functions

$$\Sigma^\infty = \{\, s \mid s : \{1, \ldots, n\} \to \{0, 1\}, 0 \leq n \leq \infty \,\}$$

is also an ω-algebraic dcpo under the extension order

$$s \sqsubseteq t \Leftrightarrow |s| \leq |t| \ \&\ (\,\forall\, 1 \leq i \leq |s|\,)\ s(i) = t(i),$$

where $|s|$ is written for the cardinality of $\mathrm{dom}(s)$. The supremum of a directed set $S \subseteq \Sigma^\infty$ is $\bigcup S$, while the approximation relation is

$$s \ll t \Leftrightarrow s \sqsubseteq t \ \&\ |s| < \infty.$$

The extension order in this special case is usually called the *prefix* order. The elements $s \in \Sigma^\infty$ are called *strings* over $\{0, 1\}$. The quantity $|s|$ is called the *length* of a string s. The *empty string* ε is the unique string with length zero. It is the least element $\perp$ of Σ^∞.

Example 4. If X is a locally compact Hausdorff space, then its *upper space*

$$\mathbf{U}X = \{\emptyset \neq K \subseteq X : K \textit{ is compact}\}$$

ordered under reverse inclusion

$$A \sqsubseteq B \Leftrightarrow B \subseteq A$$

is a continuous dcpo. The supremum of a directed set $S \subseteq \mathbf{U}X$ is $\bigcap S$ and the approximation relation is $A \ll B \Leftrightarrow B \subseteq \mathrm{int}(A)$.

Example 5. Given a metric space (X, d), the *formal ball model* [2]

$$\mathbf{B}X = X \times [0, \infty)$$

is a poset when ordered via

$$(x, r) \sqsubseteq (y, s) \Leftrightarrow d(x, y) \leq r - s.$$

The approximation relation is characterized by

$$(x, r) \ll (y, s) \Leftrightarrow d(x, y) < r - s.$$

The poset $\mathbf{B}X$ is continuous. However, $\mathbf{B}X$ is a dcpo iff the metric d is complete. In addition, $\mathbf{B}X$ has a countable basis iff X is a separable metric space.

3 Measurement

The set $[0, \infty)^*$ is the domain of nonnegative reals in their opposite order.

Definition 9. A Scott continuous map $\mu : D \to [0, \infty)^*$ on a continuous dcpo D *induces the Scott topology near* $X \subseteq D$ if for all Scott open sets $U \subseteq D$ and for any $x \in X$,

$$x \in U \Rightarrow (\exists\, \varepsilon > 0)\; x \in \mu_\varepsilon(x) \subseteq U,$$

where $\mu_\varepsilon(x) = \{y \in D : y \sqsubseteq x \;\&\; |\mu x - \mu y| < \varepsilon\}$. This is written $\mu \to \sigma_X$.

Definition 10. A *measurement* on a domain D is a Scott continuous mapping $\mu : D \to [0, \infty)^*$ with $\mu \to \sigma_{\ker \mu}$ where $\ker \mu = \{x \in D : \mu x = 0\}$.

The most useful properties of measurements in applications are as follows.

Proposition 2. *If D is a domain with a measurement $\mu \to \sigma_X$, then*

(i) *For all $x \in D$ and $y \in X$, $x \sqsubseteq y \;\&\; \mu x = \mu y \Rightarrow x = y$.*
(ii) *For all $x \in D$, $\mu x = 0 \Rightarrow x \in \max D$.*
(iii) *For all $x \in X$ and any sequence (x_n) in D with $x_n \sqsubseteq x$, if $\mu x_n \to \mu x$, then $\bigsqcup x_n = x$, and this supremum converges in the Scott topology.*

Proof For (i), we prove that $y \sqsubseteq x$. Let U be a Scott open set around y. Then there is $\varepsilon > 0$ with $y \in \mu_\varepsilon(y) \subseteq U$. But $x \sqsubseteq y$ and $\mu x = \mu y$ hence $x \in \mu_\varepsilon(y) \subseteq U$. Thus, every Scott open set around y also contains x, establishing $y \sqsubseteq x$. (ii) follows from (i). The proof of (iii) uses the same technique applied in (i) and may be found in [3]. □

Prop. 2 shows that measurements capture the essential characteristics of information content. For example, (i) says that comparable objects with the same amount of information are equal, (ii) says that an element with no disorder in it (no partiality) must be maximal in the information order, and (iii) says that if we measure an iterative process (x_n) as computing an object x, then it *actually does* calculate x. The common theme in each case is this: Any observation made with a measurement is a *reliable* one.

Example 6. Domains and their standard measurements.

(i) $(\mathbf{I}\mathbb{R}, \mu)$ the interval domain with the length measurement $\mu[a, b] = b - a$.
(ii) $([\mathbb{N} \rightharpoonup \mathbb{N}], \mu)$ the partial functions on the naturals with

$$\mu f = |\mathrm{dom}(f)|$$

where $|\cdot| : \mathcal{P}\omega \to [0, \infty)^*$ is the measurement on the algebraic lattice $\mathcal{P}\omega$ given by

$$|x| = 1 - \sum_{n \in x} \frac{1}{2^{n+1}}.$$

(iii) $(\Sigma^\infty, 1/2^{|\cdot|})$ the Cantor set model where $|\cdot| : \Sigma^\infty \to [0, \infty]$ is the length of a string.
(iv) $(\mathbf{U}X, \mathrm{diam})$ the upper space of a locally compact metric space (X, d) with

$$\mathrm{diam}\, K = \sup\{d(x, y) : x, y \in K\}.$$

(v) $(\mathbf{B}X, \pi)$ the formal ball model of a complete metric space (X, d) with $\pi(x, r) = r$.

In each example above, we have a measurement $\mu : D \to [0, \infty)^*$ on a domain with $\ker \mu = \max D$. In all cases except (iv), we also have $\mu \to \sigma_D$. In general, there are existence theorems [3] for countably based domains, which show that measurements usually exist. However, the value of the idea lies not in knowing that they exist abstractly, but in knowing that *particular* mappings, like the ones in the last example, are measurements.

4 Fixed Points of Nonmonotonic Maps

Definition 11. A *splitting* on a poset P is a function $s : P \to P$ with $x \sqsubseteq s(x)$ for all $x \in P$.

Proposition 3. *Let D be a domain with a measurement $\mu \to \sigma_D$. If $I \subseteq D$ is closed under directed suprema and $s : I \to I$ is a splitting whose measure*

$$\mu \circ s : I \to [0, \infty)^*$$

is Scott continuous between dcpo's, then

$$(\forall x \in I) \bigsqcup_{n \geq 0} s^n(x) \text{ is a fixed point of } s.$$

Moreover, the set of fixed points $\mathrm{fix}(s) = \{x \in I : s(x) = x\}$ *is a dcpo.*

Proof Let $x \in I$. By induction, $(s^n(x))$ is an increasing sequence in I. The set I is closed under directed suprema hence $\bigsqcup_{n\geq 0} s^n(x) \in I$. Because s is a splitting, $\bigsqcup_{n\geq 0} s^n(x) \sqsubseteq s(\bigsqcup_{n\geq 0} s^n(x))$, while the fact that $\mu \circ s$ and μ are both Scott continuous allows us to compute

$$\mu s(\bigsqcup_{n\geq 0} s^n(x)) = \lim_{n\to\infty} \mu s^{n+1}(x) = \mu(\bigsqcup_{n\geq 0} s^n(x)).$$

By Prop 2, however, two comparable elements whose measures agree must in fact be equal. Hence,

$$s(\bigsqcup_{n\geq 0} s^n(x)) = \bigsqcup_{n\geq 0} s^n(x).$$

To show that $\mathrm{fix}(s)$ is a dcpo one need only prove closure under suprema of sequences because $\mu \to \sigma_D$ [3]. The proof for sequences, however, uses the very same methods employed above and is entirely trivial. □

Example 7. Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous map on the real line. Denote by $C(f)$ the subset of $\mathbf{I}\mathbb{R}$ where f changes sign, that is,

$$C(f) = \{[a, b] : f(a) \cdot f(b) \leq 0\}.$$

The continuity of f ensures that this set is closed under directed suprema, and the mapping

$$\mathrm{split}_f : C(f) \to C(f)$$

given by

$$\mathrm{split}_f[a, b] = \begin{cases} \mathrm{left}[a, b] & \text{if } \mathrm{left}[a, b] \in C(f); \\ \mathrm{right}[a, b] & \text{otherwise,} \end{cases}$$

is a splitting where $\mathrm{left}[a, b] = [a, (a + b)/2]$ and $\mathrm{right}[a, b] = [(a + b)/2, b]$. The measure of this mapping

$$\mu\, \mathrm{split}_f[a, b] = \frac{\mu[a, b]}{2}$$

is Scott continuous, so Proposition 3 implies that

$$\bigsqcup_{n\geq 0} \mathrm{split}_f^n[a, b] \in \mathrm{fix}(\mathrm{split}_f).$$

However, $\mathrm{fix}(\mathrm{split}_f) = \{[r] : f(r) = 0\}$, which means that iterating split_f is a scheme for calculating a solution of the equation $f(x) = 0$. This numerical technique is called *the bisection method.*

The major fixed point technique in classical domain theory, the Scott fixed point theorem, cannot be used to establish the correctness of the bisection method: split_f is only monotone in computationally irrelevant cases.

Proposition 4. *For a continuous selfmap $f : \mathbb{R} \to \mathbb{R}$ which has at least one zero, the following are equivalent:*

(i) *The map* split_f *is monotone.*
(ii) *The map f has a unique zero r and*

$$C(f) = \{[a, r] : a \leq r\} \cup \{[r, b] : r \leq b\}.$$

Proof We prove (i) $\Rightarrow$ (ii). Let $\alpha < \beta$ be two distinct roots of f. Then by monotonicity of split_f,

$$\mathrm{split}_f^n[\alpha, \beta] \sqsubseteq \mathrm{split}_f[\beta] = [\beta],$$

for all $n \geq 0$. Then $[\alpha] = \bigsqcup \mathrm{split}_f^n[\alpha, \beta] \sqsubseteq [\beta]$, which proves $\alpha = \beta$. Thus, f has a unique zero r.

Now let $[a, b] \in C(f)$ with $a < r < b$ and set $\delta = \max\{r - a, b - r\} > 0$. Then $r - \delta \leq a < b \leq r + \delta$. By the uniqueness of r,

$$f(r - \delta) \cdot f(a) > 0 \text{ and } f(b) \cdot f(r + \delta) > 0,$$

and since $[a, b] \in C(f)$, we have $\bar{y} := [r - \delta, r + \delta] \in C(f)$. For the very same reason, $\bar{x} := [r - \delta - \delta/2, r + \delta + \delta/4] \in C(f)$. But then we have $\bar{x} \sqsubseteq \bar{y}$ and

$$\mathrm{split}_f \bar{x} = [r - \delta/8, r + \delta + \delta/4] \not\sqsubseteq [r - \delta, r] = \mathrm{split}_f \bar{y},$$

which means split_f is not monotone if f changes sign on an interval which contains r in its interior. □

That is, if split_f is monotone, then in order to calculate the solution r of $f(x) = 0$ using the bisection method, we must first *know* the solution r.

Example 8. A function $f : [a, b] \to \mathbb{R}$ is *unimodal* if it has a maximum value assumed at a unique point $x^* \in [a, b]$ such that

(i) f is strictly increasing on $[a, x^*]$, and
(ii) f is strictly decreasing on $[x^*, b]$.

Unimodal functions have the important property that

$$x_1 < x_2 \Rightarrow \begin{cases} x_1 \leq x^* \leq b \text{ if } f(x_1) < f(x_2), \\ a \leq x^* \leq x_2 \text{ otherwise.} \end{cases}$$

This observation leads to an algorithm for computing x^*. For a unimodal map $f : [a, b] \to \mathbb{R}$ with maximizer $x^\star \in [a, b]$ and a constant $1/2 < r < 1$, define a dcpo by

$$I_{x^*} = \{\bar{x} \in \mathbb{IR} : [a, b] \sqsubseteq \bar{x} \sqsubseteq [x^*]\},$$

and a splitting by

$$\max_f : I_{x^*} \to I_{x^*}$$

$$\max_f[a, b] = \begin{cases} [l(a, b), b] \text{ if } f(l(a, b)) < f(r(a, b)); \\ [a, r(a, b)] \text{ otherwise,} \end{cases}$$

where $l(a,b) = (b-a)(1-r)+a$ and $r(a,b) = (b-a)r+a$. The measure of $\max_f$ is Scott continuous since $\mu \max_f(\bar{x}) = r \cdot \mu(\bar{x})$, for all $\bar{x} \in I_{x^*}$. By Proposition 3,

$$\bigsqcup_{n \geq 0} \max_f^n(\bar{x}) \in \mathrm{fix}(\max_f),$$

for any $\bar{x} \in I_{x^*}$. However, any fixed point of $\max_f$ has measure zero, and the only element of I_{x^*} with measure zero is $[x^*]$. Thus, $\bigsqcup \max_f^n[a,b] = [x^*]$, which means that iterating $\max_f$ yields a method for calculating x^*. This technique is called the *r-section search.*

Finally, observe that $\max_f$ is not monotone. Let -1 $< \alpha <$ 1 and $f(x) = 1-x^2$. The function f is unimodal on any compact interval. Since $\max_f$[-1, 1] = [-1, $2r-1$], we see that

$$\begin{aligned} \max_f[\text{-}1,1] \sqsubseteq \max_f[\alpha,1] &\Rightarrow 1 \leq 2r-1 \text{ or } r(\alpha,1) \leq 2r-1 \\ &\Rightarrow 1 \leq r \text{ or } \alpha+1 \leq r(\alpha+1) \\ &\Rightarrow r \geq 1, \end{aligned}$$

which contradicts $r < 1$. Thus, for no value of r is the algorithm monotone.

As further evidence of its applicability, notice that Prop. 3 also implies the Scott fixed point theorem for domains with measurements $\mu \to \sigma_D$.

Example 9. If $f : D \to D$ is a Scott continuous map on a domain D with a measurement $\mu \to \sigma_D$, then we consider its restriction to the set of points where it improves

$$I(f) = \{x \in D : x \sqsubseteq f(x)\}.$$

This evidently yields a splitting $f : I(f) \to I(f)$ on a dcpo with continuous measure. By Proposition 3,

$$(\forall x \in I(f)) \bigsqcup_{n \geq 0} f^n(x) \text{ is a fixed point of } f.$$

For instance, if D is ω-continuous with basis $\{b_n : n \in \mathbb{N}\}$, then

$$\mu x = |\{n : b_n \ll x\}|$$

defines a measurement $\mu \to \sigma_D$. Notice, however, that with this construction we normally have $\ker \mu = \emptyset$.

5 Unique Fixed Points of Monotonic Maps

In the last section, we saw that measurement can be used to generalize the Scott fixed point theorem so as to include important nonmonotonic processes. Now we see that in can improve upon it for monotone maps as well, by giving a technique that guarantees *unique* fixed points.

Definition 12. Let D be a continuous dcpo with a measurement μ. A monotone map $f : D \to D$ is a *contraction* if there is a constant $c < 1$ with

$$\mu f(x) \leq c \cdot \mu x$$

for all $x \in D$.

Theorem 1. *Let D be a domain with a measurement μ such that*

$$(\,\forall\, x, y \in \ker \mu\,)(\,\exists\, z \in D\,)\; z \sqsubseteq x, y.$$

If $f : D \to D$ is a contraction and there is a point $x \in D$ with $x \sqsubseteq f(x)$, then

$$x^\star = \bigsqcup_{n \geq 0} f^n(x) \in \max D$$

is the unique fixed point of f on D. Furthermore, $x^\star$ is an attractor in two different senses:

(i) *For all $x \in \ker \mu$, $f^n(x) \to x^\star$ in the Scott topology on $\ker \mu$, and*
(ii) *For all $x \sqsubseteq x^\star$, $\bigsqcup_{n \geq 0} f^n(x) = x^\star$, and this supremum is a limit in the Scott topology on D.*

Proof First, for any $x \in D$ and any $n \geq 0$, $\mu f^n(x) \leq c^n \mu x$, as is easy to prove by induction. Given a point $x \sqsubseteq f(x)$, the monotonicity of f implies the sequence $(f^n(x))$ is increasing, while the continuity of μ allows us to compute

$$\mu(\bigsqcup f^n(x)) = \lim_{n \to \infty} \mu f^n(x) \leq \lim_{n \to \infty} c^n \mu x = 0.$$

Hence, $x^\star = \bigsqcup f^n(x) \in \ker \mu \subseteq \max D$. But the monotonicity of f also gives $x^\star \sqsubseteq f(x^\star)$. Hence, $x^\star = f(x^\star)$ is a fixed point of f. We will prove its uniqueness after (ii).

For (ii), let $x \sqsubseteq x^\star$. By the monotonicity of f,

$$(\forall n \geq 0)\; f^n(x) \sqsubseteq f(x^\star) = x^\star,$$

and since $\lim \mu f^n(x) = \mu x^\star = 0$, the fact that μ is a measurement yields

$$\bigsqcup_{n \geq 0} f^n(x) = x^\star.$$

Now let $x_\star$ be any fixed point of f. Then $x_\star \in \ker \mu$ so there is $z \in D$ with $z \sqsubseteq x_\star, x^\star$. By (ii), $\bigsqcup f^n(z) = x_\star = x^\star$. Thus, the fixed point $x^\star$ is unique.

For (i), let $x \in \ker \mu$. Then $f^n(x) \in \ker \mu$ for all $n \geq 0$. In addition, there is an $a \sqsubseteq x, x^\star$, so $f^n(a) \sqsubseteq f^n(x), x^\star$. Now let U be a Scott open set around x^*. Because μ is a measurement,

$$(\exists\, \varepsilon > 0)\; x^\star \in \mu_\varepsilon(x^\star) \subseteq U.$$

Since $\mu f^n(a) \to 0$, all but a finite number of the $f^n(a)$ are in U. But U is an upper set, so the same is true of the $f^n(x)$. Hence, $f^n(x) \to x^\star$, in the Scott topology on $\ker \mu$. □

When a domain has a least element, the last result is easier to state.

Corollary 1. *Let D be a domain with least element $\perp$ and measurement μ. If $f : D \to D$ is a contraction, then*

$$x^\star = \bigsqcup_{n \geq 0} f^n(\perp) \in \max D$$

is the unique fixed point of f on D. In addition, the other conclusions of Theorem 1 hold as well.

All the domains that we have considered in this paper have the property that $(\forall x, y \in D)(\exists z \in D)\, z \sqsubseteq x, y$, and so Theorem 1 can be applied to them.

Example 10. Let $f : X \to X$ be a contraction on a complete metric space X with Lipschitz constant $c < 1$. The mapping $f : X \to X$ extends to a monotone map on the formal ball model $\bar{f} : \mathbf{B}X \to \mathbf{B}X$ given by

$$\bar{f}(x, r) = (fx, c \cdot r),$$

which satisfies

$$\pi \bar{f}(x, r) = c \cdot \pi(x, r),$$

where $\pi : \mathbf{B}X \to [0, \infty)^*$ is the standard measurement on $\mathbf{B}X$, $\pi(x, r) = r$. Now choose r so that $(x, r) \sqsubseteq \bar{f}(x, r)$. By Theorem 1, $\bar{f}$ has a unique attractor which implies that f does also because $X \simeq \ker \pi$.

We can also use the upper space $(\mathbf{U}X, \mathrm{diam})$ to prove the Banach contraction theorem for compact metric spaces by applying the technique of the last example.

Example 11. Consider the well-known functional

$$\phi : [\mathbb{N} \rightharpoonup \mathbb{N}] \to [\mathbb{N} \rightharpoonup \mathbb{N}]$$

$$\phi(f)(k) = \begin{cases} 1 & \text{if } k = 0, \\ kf(k-1) & \text{if } k \geq 1 \ \&\ k - 1 \in \mathrm{dom}\, f. \end{cases}$$

which is easily seen to be monotone. Applying $\mu : [\mathbb{N} \rightharpoonup \mathbb{N}] \to [0, \infty)^*$, we compute

$$\begin{aligned} \mu\phi(f) &= |\mathrm{dom}(\phi(f))| \\ &= 1 - \sum_{k \in \mathrm{dom}(\phi(f))} \frac{1}{2^{k+1}} \\ &= 1 - \left(\frac{1}{2^{0+1}} + \sum_{k-1 \in \mathrm{dom}(f)} \frac{1}{2^{k+1}} \right) \\ &= 1 - \left(\frac{1}{2} + \sum_{k \in \mathrm{dom}(f)} \frac{1}{2^{k+2}} \right) \\ &= \frac{1}{2} \left(1 - \sum_{k \in \mathrm{dom}(f)} \frac{1}{2^{k+1}} \right) \\ &= \frac{\mu f}{2} \end{aligned}$$

which means ϕ is a contraction on the domain $[\mathbb{N} \rightharpoonup \mathbb{N}]$. By the contraction principle,

$$\bigsqcup_{n \in \mathbb{N}} \phi^n(\perp) = \mathrm{fac}$$

is the unique fixed point of ϕ on $[\mathbb{N} \rightharpoonup \mathbb{N}]$, where $\perp$ is the function defined nowhere.

One wonders here about the potential for replacing metric space semantics with an approach based on measurement and contractions.

6 Closing Remarks

There are many ideas left from the present discussion on measurement. Among the most fundamental are the μ topology, the study of the topological structure of $\ker \mu$, a discussion of how one extends measurements to higher order domains, and the informatic derivative (the derivative of a map on a domain with respect to a measurement). All of this can be found on the author's webpage in [3].

References

1. S. Abramsky and A. Jung. *Domain Theory.* In S. Abramsky, D. M. Gabbay, T. S. E. Maibaum, editors, Handbook of Logic in Computer Science, vol. III. Oxford University Press, 1994
2. A. Edalat and R. Heckmann. *A Computational Model for Metric Spaces.* Theoretical Computer Science 193 (1998) 53–73 .
3. K. Martin. *A foundation for computation.* Ph.D. thesis, Department of Mathematics, Tulane University, 2000.

Graph Transformation as a Conceptual and Formal Framework for System Modeling and Model Evolution

Gregor Engels and Reiko Heckel

University of Paderborn, Dept. of Computer Science, D 33095 Paderborn, Germany
{engels|reiko}@upb.de

Abstract. Distributed software systems are typically built according to a three layer conceptual structure: Objects on the lowest layer are clustered by components on the second layer, which themselves are located at nodes of a computer network on the third layer. Orthogonal to these three layers, an instance level and a type or schema level are distinguished when modeling these systems. Accordingly, the changes a system experiences during its lifetime can be classified as the system's dynamic behavior on the instance level and as the evolution of the system on the schema level. This paper shows how concepts from the area of graph transformation can be applied to provide a conceptual and formal framework for describing the structural and behavioral aspects of such systems.

Keywords: typed graph transformation, hierarchical graphs, system modeling, model evolution

1 Introduction

The structure and characteristics of software systems have changed dramatically during the last four decades. Starting with small programs in the 1960s and continuing with large monolithic systems in the 1970s, the 1980s faced a development towards hierarchically structured subsystems or modules. Today, software systems represent complex, often dynamic networks of interacting components or agents. In order to adapt to changing requirements and application contexts, and supported by modern programming language features (as, e.g., Java RMI [36]), middleware standards (like CORBA or DCOM [28,7]), and world-wide connectivity, these components may evolve over time, and they may be down-loaded and linked together while the system is executing.

Beside the development from monolithic and static towards distributed and mobile applications, the process of software development has also changed. Today's software development teams consist of developers with different expertise, skills, and responsibilities. Due to resource management, organizational, or administrative reasons, the development process itself may be distributed. Moreover, since complex systems can no longer be built from scratch, an incremental

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 127–150, 2000.
© Springer-Verlag Berlin Heidelberg 2000

software development process is employed which is structured in several phases like, for example, a requirements analysis, design, and implementation phase (cf. [24]).

A major concern for the success of such a distributed, incremental process is the maintenance of consistency in different dimensions: for instance, between the behavior of different components of the system, between the concepts used by different development teams in the project, or between the artifacts of different phases of development.

An important means for achieving consistency in all dimensions is to build a model. It can, for instance, facilitate formal verification of component interaction, form the basis for communication between teams, and serve as project documentation where all relevant decisions are documented in order to trace design choices between different phases of development. In particular, a *requirements specification document* is indispensable in which the functionality of the software system to be built is stated. This document may represent a contract between the customer and the software development team as well as a contract within the team between software engineers being responsible for the analysis and design tasks and the programmers being responsible for a correct and efficient implementation.

Thus, the model has to be written in a language which is intuitively and easily understandable by customers, in general not being computer scientists, as well as by software engineers and programmers, hopefully being computer scientists. Particularly, this means that the used *specification* or *modeling language* has to provide language features which are on an appropriate abstraction level. Thus, the language has to offer support to abstract from programming or even machine-dependent details on the one hand and from unimportant real-world details on the other hand (cf. Fig. 1).

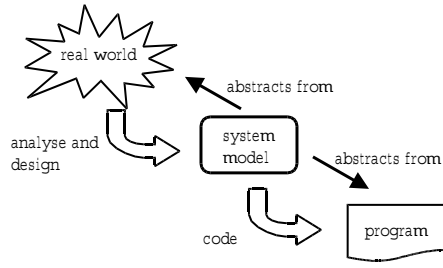


Fig. 1. Role of a system model.

Often, informal diagrammatic modeling languages like the Unified Modeling Language (UML) [27] are considered the first choice from this point of view: They are visual and apparently easy to understand and, due to the use of various diagrams and special-purpose notations for different aspects, they provide

good support for high-level specifications. However, as we can judge from our own experience in teaching and modeling in UML, the language still lacks a clear underlying conceptual model of the systems to be modeled [20]. But, an agreement on such a conceptual model has to precede the definition of syntax and semantics of a language, as only on this base appropriate language features, a precise semantics, and adequate pragmatic guidelines on how to use the language can be given.

Therefore, the aim of this paper is twofold. First, we want to contribute to the *understanding of the conceptual model* underlying today's software systems. This shall be done by presenting a typical example of a mobile, distributed application and by identifying three relevant dimensions. Second, we shall provide a *formalization of the conceptual model* using concepts and results from the theory of graph transformation systems. The basic idea is to model the system's states as graphs, and to specify its dynamic change by transformation rules.

In the theory of semantics, various formalisms have been developed for the operational specification of concurrent, distributed and mobile systems. Most of them are based on a notion of state much simpler than graphs. Among the numerous term-based approaches we only mention process calculi like [42,3,34,43] building on labeled transition systems and the Structured Operational Semantics (SOS) paradigm [44], and approaches based on rewriting systems like [41,4]. In Petri net-based approaches [46] the states of a system are modeled as sets or multi-sets. These formalisms have been quite successful in modeling relevant aspects of software systems, and they have developed a large body of theory which can aid the developer in structuring, understanding, and verifying her specifications.

Still, we believe that graphs and graph transformation are indispensable means for providing an operational, conceptual and formal framework for today's software development. First, the states of object-oriented, distributed, and mobile systems are most naturally represented as graphs modeling, e.g., object structures, software architectures, network topologies, etc. Then, for describing changes to these states like the insertion or deletion of objects or links, architectural or network reconfiguration, graphs have to be manipulated. Second, in diagrammatic modeling languages, the abstract syntax of an individual model is usually represented as a graph instead of a term or tree as in (textual) programming languages. Thus, for translating one diagram language into another, for defining the semantics of diagrams by reducing them to a normal form or by animating them, graph-based techniques are needed where, in the case of textual languages, term rewrite systems or SOS specifications may be used.

In this paper, we will be interested in the first issue of interpreting graphs as states and graph transformation systems as "programs". The second idea of graphs as representing diagrams and graph transformation systems defining their operational semantics is dealt with, for example, in [8,19]. In the next section, we will discuss a simple application scenario of a distributed and mobile system. This will serve for identifying the relevant dimensions of a conceptual framework for expressing structural and behavioral aspects as well as the evolution of today's

software systems. Based on these investigations, in Sect. 3 we will use graph transformation systems to formalize our conceptual framework making explicit the consistency and orthogonality of its different dimensions. We close with a summary and a discussion of further aspects of software development which are not covered in the paper.

2 A Conceptual Model for Distributed and Mobile Application

In this section, we discuss and analyze a concrete application scenario of a distributed system with mobile hardware components in order to identify the relevant concepts and dimensions in modeling such systems. This conceptual model shall provide the basis for the formal approach developed in Sect. 3.

Our application scenario is (a simplified version of) a distributed system from the financial domain which uses Java cards (i.e., smartcards with a microprocessor supporting the JavaCard runtime environment [35]) to handle financial transactions (like the paying of bills) which require the coordination of services from different providers (e.g., a shop and a bank). A customer possessing such a smartcard may for instance, after purchasing some goods in a shop, use this card to download a corresponding bill from a cash box. Later on, she may pay her bills at a banking terminal. This terminal is one out of several hundreds, connected via an intranet connection to the mainframe of the bank. Thus, the scenario involves four different types of hardware components: cash boxes, smartcards, banking terminals, and mainframes (cf. Fig. 2). Each hardware component has a processor and memory, and can host and execute software components.

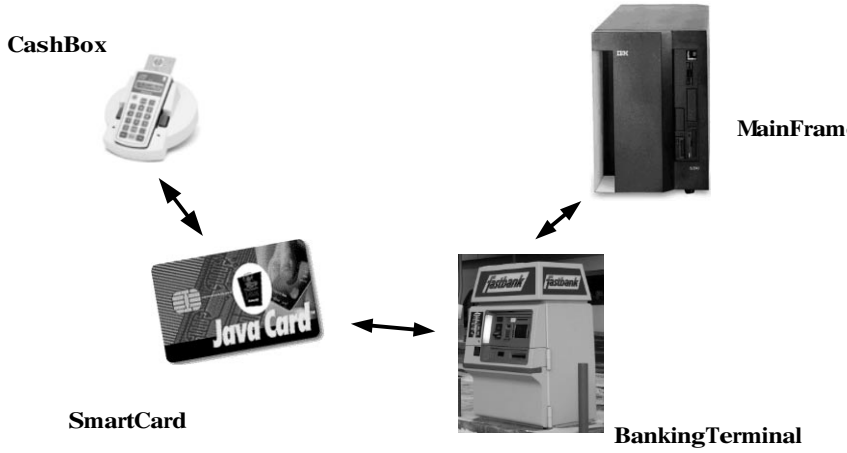


Fig. 2. Sample application scenario.

2.1 Hierarchical Structures

Fig. 3 shows a snapshot of the hardware architecture of the scenario on a more abstract level. The diagram depicts five hardware components where the SmartCard component is linked to one of two BankingTerminal components which are both connected to a MainFrame. The diagram is shown in a UML-like notation [27], where instances i of class C are represented as i:C (or simply :C if the identity is not of interest). The boxes are an iconic notation for hardware components (called *nodes*) as it is known from UML deployment diagrams.

This hardware architecture forms the base layer of any running software system. Each node is equipped with some software components, which interact with each other locally (at one node) or remotely (between different nodes). These software components form the second layer of a software system.

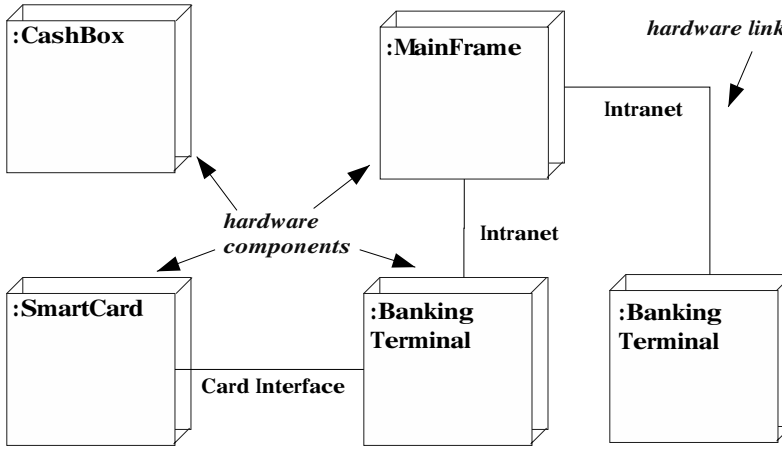


Fig. 3. Hardware architecture of the application scenario (snapshot).

The allocation of software components to nodes can be described by UML deployment diagrams. A sample diagram is shown in Fig. 4 where the SmartCard hosts a BillCard component responsible for accepting a bill issued by the Billing component of a CashBox as well as for storing this bill and transferring it to a BankingTerminal for payment. Notice that software components may be temporarily linked even if they reside on different nodes. A node may also host several different software components as, for instance, the SmartCard which carries a separate AuthCard component responsible for the authentication of the card owner.

From a technical point of view, these diagrams are graphs with labeled (and attributed) vertices and edges. The deployment diagram in Fig. 4 can even be

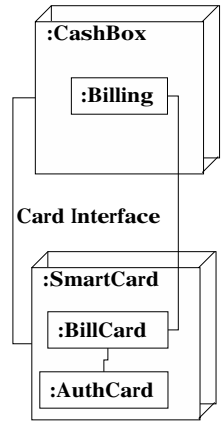


Fig. 4. Deployment diagram.

viewed as a kind of *hierarchical graph* where vertices representing hardware nodes contain vertices representing software components. Due to the links between software components running on different nodes, however, this hierarchy is not strict, as sublayers of different subgraphs may be directly interconnected.

On top of the hardware architecture and component layers, a third layer can be identified. These are the *problem-domain objects* (the objects doing the actual work) within software components. Fig. 5 provides an extended deployment diagram which, in addition, depicts problem-domain objects like the :Bill and :Customer located within the Billing component.

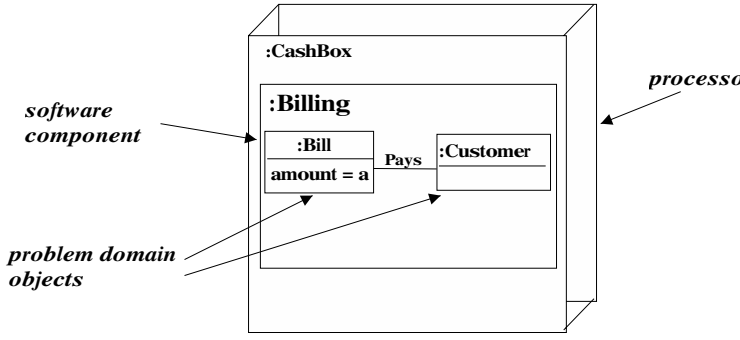


Fig. 5. Deployment diagram extended by problem-domain objects.

That is, our conceptual framework for system modeling shall be based on hierarchical graphs where at least three layers have to be distinguished (cf. Fig. 11). Moreover, we often encounter hierarchical structures even within any of the three layers of objects, software components, and hardware architecture. This multi-layered *hierarchical graph structure* forms the first dimension of our conceptual framework.

2.2 Typed Structures

All diagrams discussed so far have shown individual snapshots of the application scenario. In the world of programming, this corresponds to a state in the execution of a program which is specified in the program text by means of variables and types. Analogously, in modeling we also distinguish between an instance and a type level and demand that the instance level is consistent with the type level.

In the layer of problem-domain objects, the type level is provided by a class diagram as depicted in Fig. 6. It defines three different classes and their associations where the multiplicity is restricted by appropriate cardinality constraints. Classes are described by their names and by attribute definitions, whereas instances of a class will have concrete values for these attributes.

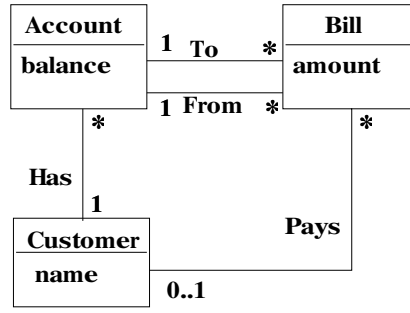


Fig. 6. Problem-domain class diagram.

Analogously, the allowed software and hardware architectures on the instance level may be determined by appropriate type definitions. Type definitions for software architectures, also called *architectural styles* [51], determine a set of admissible concrete architectures. Fig. 7 gives an example for the definition of hardware architectural style using a hardware diagram on the type level. The notation is similar to that of the class diagram in Fig. 6 but for the use of the *node* icon. The xor-constraint ensures that a SmartCard instance is either connected to a CashBox or to a BankingTerminal.

Hence, beside the hierarchical structure of the model we have identified a second, orthogonal dimension of *typing*. The states of our conceptual model are

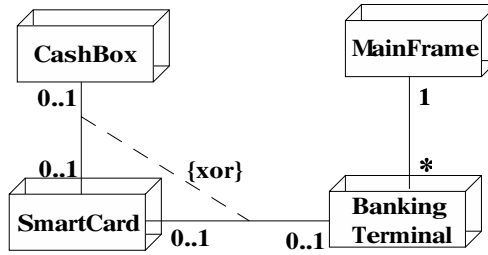


Fig. 7. Hardware architecture (type level).

therefore multi-layered, non-strict hierarchical graphs typed over correspondingly structured type graphs.

2.3 Instance Level Dynamics

So far, we have only dealt with the structural aspects of our model. In order to provide semantics for distributed and mobile applications with dynamic reconfiguration we have to specify operations for transforming this structure.

Dynamic change on the instance level is specified by transformation rules as shown in Fig. 8. A rule consists of two object diagrams, the left- and the right-hand side, where the former specifies the situation before the operation and the latter the situation afterwards. The rule in Fig. 8 describes the operation of a customer paying a bill from his account by transferring the required amount to the specified target account. The transformation of object structures has to be consistent with the typing in the sense that the constraints on the instance level imposed by the type-level diagrams are preserved.

The object transformation shown in Fig. 8 can be seen as an abstract requirement specification of the operation payBill disregarding the constraints imposed by the architectural layer. According to our scenario, Account and Bill objects originally reside on different nodes. Therefore, the bill has to be transferred from the cash box to the banking system by means of a SmartCard. The operations required for downloading the bill onto the card are described in Fig. 9. When the card is inserted into the cash box's card reader, a hardware connection is established. This triggers, after completion of an appropriate authentication protocol, the connection of the Billing with the BillCard component. Then, the bill is stored in the BillCard component and the customer's identity is recorded by the Billing component of the cash box.

As indicated by the identities of the Customer and the Bill objects, these objects are not simply copied from one component to the other but they are actually shared after the operation. This is physically impossible, but object-oriented middle-ware like CORBA [28] supports the sharing of objects (as well as references between objects on different machines) through the concept of *proxies*, i.e., local place-holders providing access to remote objects. As conceptual

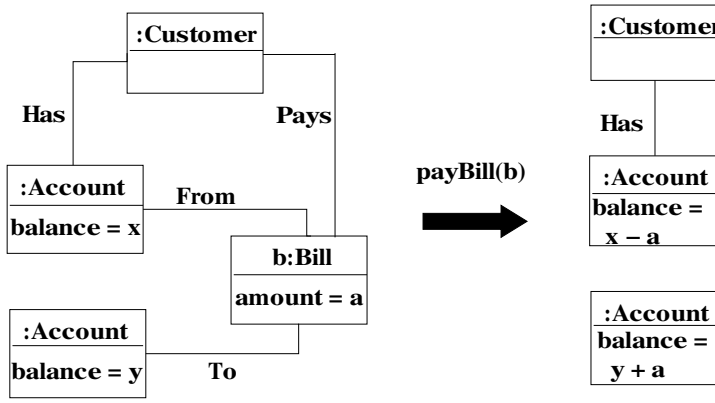


Fig. 8. Object transformation rule.

framework for the application developer, our model should provide features (at least) on the same level of abstraction as offered by state-of-the-art programming technology. Of course, in order to allow for sharing on the instance level, also on the type level it must be possible to share classes between different components types.

Notice that we have described dynamic changes within all three layers of our hierarchy and all but the first transformation are concerned with two layers at the same time. Thus, not only the states of our systems are hierarchical, but also the operations have to take care of the hierarchical structure, as they are potentially not restricted to a single layer.

2.4 Type Level Evolution

The last point to be discussed in this section is the difference between the dynamic change on the instance level as described above and the evolution of the system by changes on the type level. As systems have to be adapted to new requirements, not only their configuration may change, but also it may be necessary to introduce new types of hardware or to deploy new software components containing objects of classes which have not been known before.

In our application scenario, a new component CashCard is downloaded on the card in order to provide the additional service of using the card directly for paying bills at a cash box. A corresponding Cashing component is needed at cash boxes while banking terminals have to provide a CashCardService to transfer virtual money to the card. On the hardware level, the evolution consists in establishing a new kind of hardware connection in order to transfer bills from the cash box to the banking system via the internet. The new types are shown in the diagram of Fig. 10.

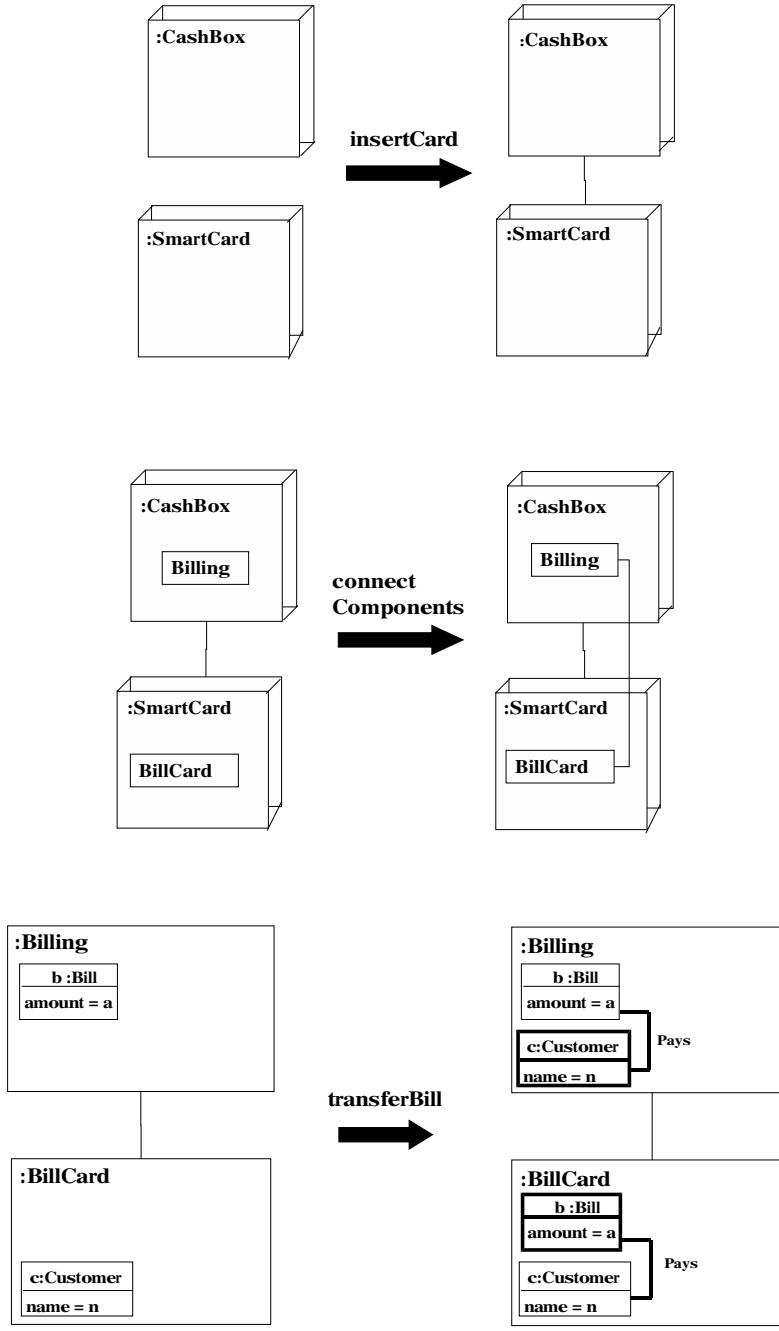


Fig. 9. Transformation of the instance level hierarchy.

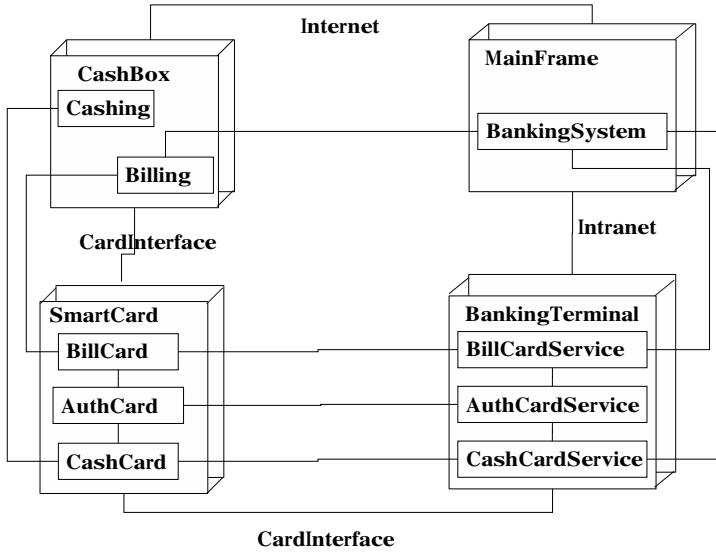


Fig. 10. Evolution of hardware and software architecture.

The overall framework is summarized in Fig. 11. It distinguishes between an *instance level* and a *type level*. Within each level, a hierarchical structure of three layers can be identified. For instance on the instance level, we have the problem-domain objects on the first layer, which are distributed over a component-based software architecture on the second layer, where software components are spread over a network of hardware components on the third layer. Orthogonally to these two dimensions of typing and hierarchical structures, we consider, as a third dimension, the dynamic change on the instance level and the evolutionary change on the type level, both in all of the three layers. The formalization and consistent integration of these three dimensions is the subject of the next section.

3 Formalizing the Conceptual Model with Graph Transformation Systems

In Sect. 2, we have analyzed the structures relevant for the modeling of distributed and mobile applications. In retrospect, in a single model we can identify the three dimensions of *typing*, dynamic and evolutionary *change*, and of the *hierarchy* of objects, software and hardware components. In this section, these three dimensions as well as their interdependencies shall be formalized.

We build upon concepts from the theory of graph transformation systems which focus on the specification of systems where states are represented as graphs and changes are specified by transformation rules (thus dynamic change is already present). Surveys on the state-of-the-art of theory and application of graph

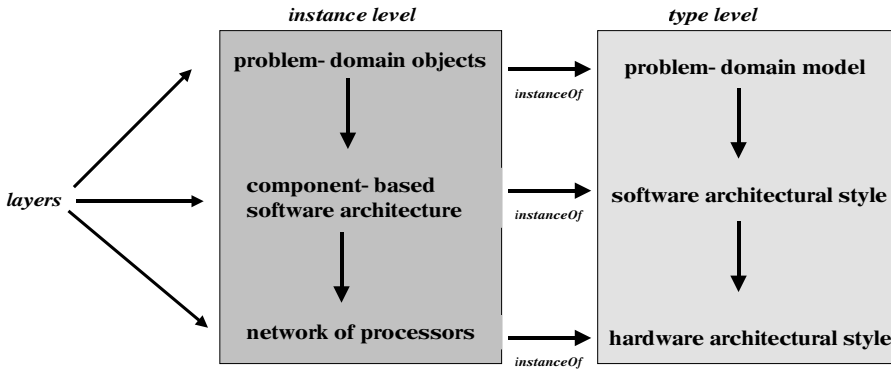


Fig. 11. Instance and type level as well as the three-layered hierarchy of system models.

transformation can be found in the three handbook volumes [47,14,16]. A good introductory text is [1].

Among the various formalizations of graph transformation, the "algebraic, Double PushOut (DPO) approach" [18,10] is one of the most successful, mainly because of its flexibility. In fact, since the basic definitions of rule and transformation are based on diagrams and constructions in a category, they can be used in a uniform way for a wide range of structures. Therefore, many results can be proved once and for all using categorical techniques [17]. This flexibility shall be exploited in the following in order to augment the original formalism [18] with several levels of typing (in the spirit of the typed DPO approach [9]), thus representing the hierarchy within a model and its evolution on the type level.

3.1 Object Dynamics as Typed Graph Transformation

In this section, we are going to provide the formal semantics of the dynamic change at the instance level conceptually discussed in Sect. 2.3. This includes the concepts of typing as described in Sect. 2.2 as well as the transformation of instances in a type-consistent way.

Graphs and typed graphs. The relation between diagrams on the type level and diagrams on the instance level is formally captured by the concept of *type* and *instance graphs* [9].

By *graphs* we mean directed unlabeled graphs $G = \langle G_V, G_E, src^G, tar^G \rangle$ with set of vertices G_V, set of edges G_E, and functions $src^G : G_E \to G_V$ and $tar^G : G_E \to G_V$ associating with each edge its source and target vertex. A graph homomorphism $f : G \to H$ is a pair of functions $\langle f_V : G_V \to H_V, f_E : G_E \to H_E \rangle$ preserving source and target, that is, $src^H \circ f_E = f_V \circ src^G$ and $tar^H \circ$

$f_E = f_V \circ tar^G$. The *category of graphs and graph morphisms* with composition and identities defined componentwise on the underlying functions is denoted by **Graph**.

Definition 1 (typed graphs). *Given a graph TG, called* type graph*, a TG-*typed (instance) graph *consists of a graph G together with a typing homomorphism $g : G \to TG$ associating with each vertex and edge x of G its type $g(x) = t$ in TG. In this case, we also write $x : t \in G$. A TG-typed graph morphism between two TG-typed instance graphs $\langle G, g\rangle$ and $\langle H, h\rangle$ is a graph morphism $f : G \to H$ which preserves types, that is, $h \circ f = g$. The category of all instance graphs typed over TG is denoted by $\mathbf{Graph}_{TG}$.*

$$\begin{tikzcd} G \arrow[d, "g"] \\ TG \end{tikzcd} \qquad \begin{tikzcd} G \arrow[rr, bend left, "f"] \arrow[dr, "g"'] & & H \arrow[dl, "h"] \\ & TG & \end{tikzcd}$$

Categorically speaking, $\mathbf{Graph}_{TG}$ is the *comma category* $(\mathbf{Graph} \downarrow TG)$ *of graphs over TG* (see, e.g., [40]). This observation allows to inherit categorical constructions from **Graph** to $\mathbf{Graph}_{TG}$. In particular, since the category of graphs has all limits and colimits, the comma category $\mathbf{Graph}_{TG}$ has all limits and colimits as well, and their constructions coincide in **Graph** and $\mathbf{Graph}_{TG}$ up to the additional typing information.

Example 1 (type and instance graphs). Fig. 12 shows the class diagram of Fig. 6 as a type graph, as well as a corresponding instance graph. We use the UML-like notation vertex:type for specifying instance graphs and their typing. The name of the vertex may be omitted in which case the diagram represents an isomorphism class of graphs. We do not formally deal with attributes in this paper. For approaches to the transformation of attributed graphs the reader is referred to [48,39,5].

Typed graph transformations. The DPO approach to graph transformation has originally been developed for vertex- and edge-labeled graphs [18]. Here, we present immediately the typed version [9].

According to the DPO approach, graph transformation rules (also called graph productions), are specified by pairs of injective graph morphisms $(L \xleftarrow{l} K \xrightarrow{r} R)$, called rule spans. The left-hand side L contains the items that must be present for an application of the rule, the right-hand side R those that are present afterwards, and the context graph K specifies the "gluing items", i.e., the objects which are read during application, but are not consumed. The transformation of graphs is defined by a pair of pushout diagrams, a so-called double pushout.

Definition 2 (DPO graph transformation). *Given a type graph TG, a TG-typed graph transformation rule is a pair $p : s$ consisting of a rule name p and a rule span $s = (L \xleftarrow{l} K \xrightarrow{r} R)$.*

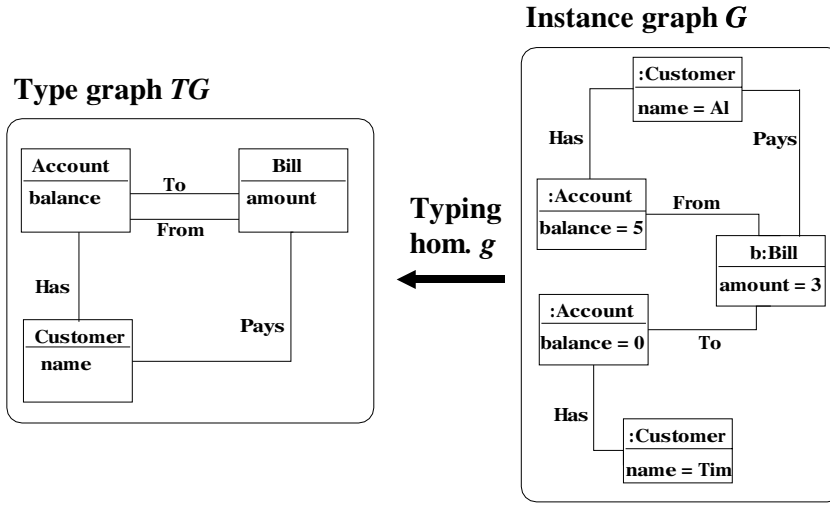


Fig. 12. Type and instance graph.

A double-pushout (DPO) diagram *o is a diagram like below where (1) and (2) are pushouts. Given a rule* $p : s$ *like above the corresponding* (direct DPO) transformation *from* G *to* H *is denoted by* $G \overset{p/o}{\Longrightarrow} H$.

$$\begin{tikzcd} L \arrow[d, "o_L"'] \arrow[dr, phantom, "(1)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "o_K"] \arrow[dr, phantom, "(2)"] & R \arrow[d, "o_R"] \\ G & D \arrow[l, "g"] \arrow[r, "h"'] & H \end{tikzcd}$$

The DPO diagram o is a categorical way of representing the occurrence of a rule in a bigger context. Assuming a simpler representation of rules, the same situation can also be expressed in a more set-theoretic way. In fact, a graph transformation rule can be represented (up to renaming) as a pair of graphs with rule name $p : L \to R$ such that the union $L \cup R$ is defined (i.e., graphs L and R live in the same name space). The span representation can be recovered as $p : L \hookleftarrow L \cap R \hookrightarrow R$. That is, given that $L \cup R$ is defined, the interface graph K, which is needed in the categorical setting for specifying the sharing between L and R, can be reconstructed and is therefore omitted.

Then, a direct transformation from G to H (with $G \cup H$ defined) using rule $p : L \to R$ is given by a graph morphism $o : L \cup R \to G \cup H$, called *occurrence*, such that

- $o(L) \subseteq G$ and $o(R) \subseteq H$, i.e., the left-hand side of the rule is embedded into the pre-state and the right-hand side into the post-state

– $o(L \setminus R) = G \setminus H$ and $o(R \setminus L) = H \setminus G$, i.e., exactly that part of G is deleted which is matched by elements of L not belonging to R and, symmetrically, that part of H has been added which is matched by elements new in R.

Like in the categorical definition, the resulting graph H is only determined up to isomorphism by the rule $p : L \to R$ and the occurrence $o_L : L \to G$.

Example 2 (typed graph transformation). Fig. 13 shows the span representation of the object transformation rule of Fig. 8 as well as its application to the instance graph in Fig. 12. Operationally speaking, the application of the rule proceeds as follows. Given the occurrence o_L of the left-hand-side in the given graph G determined by the mapping of the b:Bill vertex, the application consists of two steps: The objects of G matched by $L \setminus l(K)$ are removed which leads to the graph D without the bill. Then, the objects matched by $R \setminus r(K)$ are added to D leading to the derived graph H.

Gluing the graphs L and D over their common part K yields again the given graph G, i.e., the left-hand square (1) forms a so-called *pushout complement*. Only in this case the application is permitted. Similarly, the derived graph H is the gluing of D and R over K, which forms the right-hand side pushout square (2).

The same rewrite mechanism is used for the update of attributes: In order change an attribute value, the attribute is deleted and re-generated with the new value.

The formalization of deletion as "inverse gluing" implies that the application of a rule can be described as an embedding of $K = L \cap R$ into a context by means of the occurrence morphism $o_K : K \to D$ (cf. Fig. 13). This implies that only vertices in the image of $L \cap R$ can be merged or connected to edges in the context. These observations are reflected, respectively, in the *identification* and the *dangling condition* of the DPO approach which characterize, given a rule $p : (L \xleftarrow{l} K \xrightarrow{r} R)$ and an occurrence $o_L : L \to G$ of the left-hand side, the existence of the pushout complement (1), and hence of a direct derivation[1] $G \overset{p/o}{\Longrightarrow} H$. The *identification condition* states that objects from the left-hand side may only be identified by the match if they also belong to the interface (and are thus preserved). The *dangling condition* ensures that the structure D obtained by removing from G all objects that are to be deleted is indeed a graph, that is, no edges are left "dangling" without source or target node.

In this way, the DPO construction ensures the consistency of the transformation with the graph structure. Type consistency is ensured by the fact that the transformation rules themselves are well-typed. Together with the typing of the given graph G this induces the typing of the derived graph H.

[1] The pushout (2) always exists since category $\mathbf{Graph_{TG}}$ is cocomplete due to the cocompleteness of **Graph**.

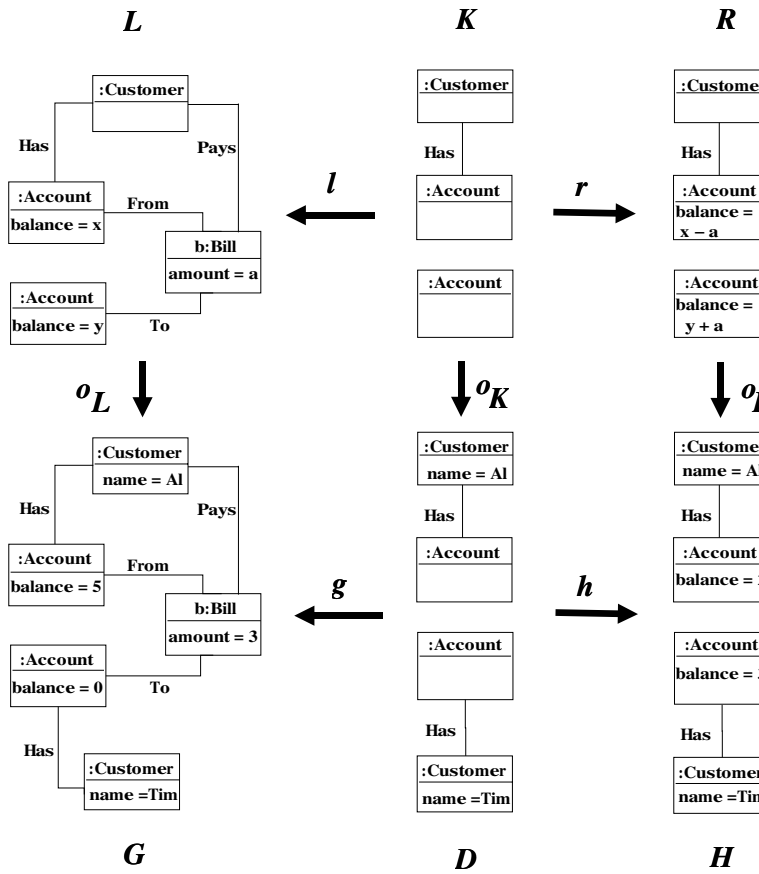


Fig. 13. DPO graph transformation step.

3.2 Hierarchical Structures as Aggregated Graphs

In Sect. 2.3, we have pointed out the requirements for modeling the dynamic change of the instance level hierarchy consisting of the object, software and hardware component layers where, in general, each of these layers may be again hierarchical. It has been observed that these hierarchies are not strict (neither on the type nor on the instance level) in the sense that vertical sharing is allowed, i.e., the refinement relation forms a directed acyclic graph (DAG) rather than a tree. In this section, we outline an approach to formalize the transformation of such structures.

Hierarchical graphs and their transformation have received much attention in the graph transformation literature. The first formal account we are aware of is [45]. Close to the conceptual model outlined in Sect. 2.1 is the approach of

distributed graph transformation [52]. Distributed graphs are hierarchical graphs with two layers only. Formally, they are diagrams in the category of graphs. That means, a graph G representing the network topology is attributed with graphs and graph morphisms modeling, respectively, the object structures contained in the local components and relationships between them. The transformation of these graphs is based on the double-pushout approach. As distributed graphs also allow sharing of objects between different components, they would be a natural candidate for formalizing our hierarchical models. However, they are restricted to hierarchies of depth two (although the construction of diagrams in the category of graphs could be iterated).

A new approach towards hierarchical graph transformation based on the DPO transformation of hypergraphs (see, e.g., [13]) which allows hierarchical graphs of arbitrary finite depth is proposed in [12]. Conceptually, hyperedges in this approach represent components which contain other hypergraphs, and the attachment vertices of a hyperedge represent ports through which the components are connected. The drawback of this approach from our point of view is its limitation to strict, tree-like hierarchies without vertical sharing.

More general concepts of hierarchical graphs allowing both vertical sharing and multiple-layered structures have been proposed, for example, in [21,6] but neither of them formally accounts for the transformation of these graphs. Summarizing, none of the approaches we are aware of satisfies all the requirements of Sect. 2 although many approaches contribute to some aspect of the overall problem.

Instead of extending the underlying formalism as it is done in most approaches, we propose to model hierarchical graphs with vertical sharing by *aggregated graphs*, i.e., graphs with distinguished aggregation edges representing the refinement of vertices. This approach has the advantage of preserving the theoretical results and semantic properties of the basic typed graph transformation approach. In particular, the concurrent semantics in terms of concurrent traces, graph processes, and event structures (see [2] for a recent survey) is relevant to our aim of developing a formal model for distributed and mobile systems.

For distinguishing between aggregation edges and ordinary edges in both type and instance graphs, we introduce an additional level of typing, the meta type level. In the *meta type graph* TG_0 in Fig. 14 on the left two kinds of edges are defined: ordinary edges and aggregation edges with a diamond head designating the super-vertex. Since there is only one node representing all kinds of vertices, TG_0 is the most general type graph serving our purpose. It could be further refined by separating the three layers of objects, software and hardware components as shown in the same figure on the right. In fact, since the graph TG_1 on the right is itself typed over TG_0, it could be used to provide another level of typing (that we omit for the sake of simplicity). An analogous approach can be found in the standardization of the UML by the OMG [27] based on the Meta Object Facility [26], a meta modeling framework which provides four levels of typing.

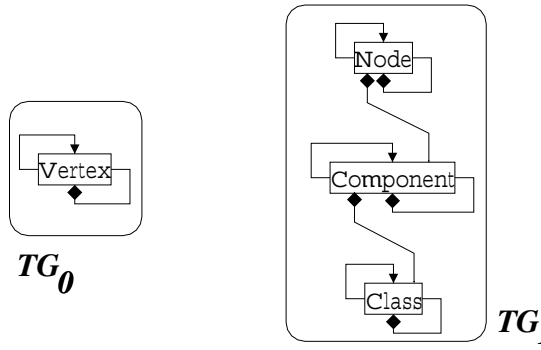


Fig. 14. Meta type graphs.

Thus, graphs typed over TG_0 represent graphs with aggregation edges. Since the genuine type level is still missing, *typed* graphs with aggregation edges are defined by replacing in Definition 1 the category of graphs and graph morphisms with the category $\mathbf{Graph}_{TG_0}$ of TG_0-typed graphs. Given a TG_0-typed graph $\langle TG, tg_0 : TG \to TG_0\rangle$ representing a type graph with aggregation edges, we build the comma category $\mathbf{Graph}_{TG_0} \downarrow \langle TG, tg_0\rangle$ of TG_0-typed graphs over $\langle TG, tg_0\rangle$. Objects of this category are graphs with two levels of typing compatible with the typing of TG as represented by the commutative diagram below on the left.

$$\begin{tikzcd} G \arrow[r, "g"] \arrow[dr, "g_0"'] & TG \arrow[d, "tg_0"] \\ & TG_0 \end{tikzcd} \qquad \begin{tikzcd} G \arrow[r, "g"'] \arrow[rr, bend left, "f"] \arrow[dr, "g_0"'] & TG \arrow[d, "tg_0"] & H \arrow[l, "h"] \arrow[dl, "h_0"] \\ & TG_0 & \end{tikzcd}$$

A morphism of $\langle TG, tg_0\rangle$-typed graphs $\langle G, g\rangle$ and $\langle H, h\rangle$ is a graph morphism f such that the diagram above on the right commutes. In particular, notice that both the meta type graph TG_0 and the type graph $\langle TG, tg_0\rangle$ remain fixed, i.e., the only variation allowed is on the level of instances.

Example 3 (typed graph with aggregation). Fig. 15 shows a type graph with aggregation edges which is itself an instance of the meta type graph TG_1 in Fig. 14. Notice the vertical sharing of the classes Customer and Bill between the Billing and BillCard components as well as the link between the two components contained in different nodes.

The transformation of $\langle TG, tg_0\rangle$-typed graphs is defined by replacing in Definition 2 the category $\mathbf{Graph}_{TG}$ for a given type graph TG with the category $\mathbf{Graph}_{TG_0} \downarrow \langle TG, tg_0\rangle$ for a given TG_0-typed type graph $\langle TG, tg_0\rangle$. Due to the notion of morphism, the transformation is restricted to the instance level while the type level remains static. Notice again that the use of a comma category does not only allow us to inherit the categorical structure relevant for the

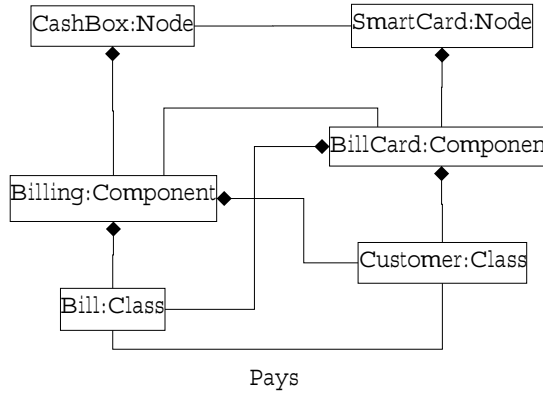


Fig. 15. A type graph with aggregation edges.

definition of transformation, but also to reestablish most theoretical results of double-pushout graph transformation.

Example 4 (graph transformation rule with aggregation). A graph transformation rule conforming to the type graph of Fig. 15 is shown in Fig. 16. It represents the formal counterpart of the lower rule transferBill in Fig. 9 (the context graph is omitted). Here, the sharing of Bill and Customer between the Billing and the BillCard components, which has been noted already in the type graph, occurs on the level of instance graphs in the right-hand side of the rule rule.

Although the concept of meta typing allows us to represent graphs with aggregation edges, additional constraints are required in order to exclude meaningless structures. In particular, we have to require the acyclicity of aggregation edges in the instance level graphs. Thus, given a TG_0-typed type graph $\langle TG, tg_0 \rangle$, an *aggregated graph* is a $\langle TG, tg_0 \rangle$-typed instance graph such that the aggregation edges form a directed acyclic graph. Further, an *aggregated graph transformation rule* is a rule in $\mathbf{Graph}_{TG_0} \downarrow \langle TG, tg_0 \rangle$ which does not introduce a new

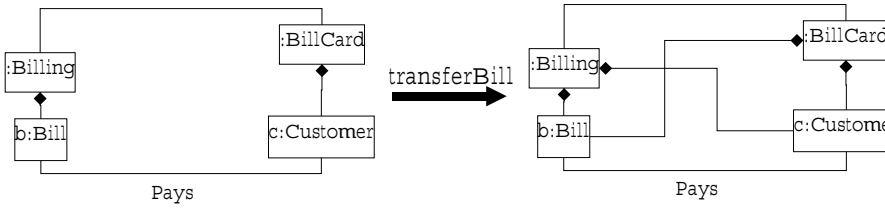


Fig. 16. Graph transformation rule with aggregation.

aggregation edge between two already existing vertices. This is enough to show that the application of a rule on an aggregated graph yields again an aggregated graph (i.e., it does not create cycles of aggregation edges), thus ensuring the consistency between dynamic change and hierarchical structure.

Of course, other constraints may be added, for example, about the connections allowed between graphs within different nodes or between different layers of the hierarchy (see, e.g., [21]). Such constraints, however, are highly application dependent and they should not be part of the basic formalism. Instead, a constraint language is required for specifying such constraints and verifying the consistency of rules. The most prominent logical approach for specifying graph properties [11] is based on second-order monadic logic. In contrast to the first-order case, this allows the specification of path properties like the acyclicity constraints mentioned above. A less powerful formalism for expressing integrity constraints which is based on a graphical notation has been introduced in [33].

3.3 Model Evolution through Type Level Transformation

In the previous section, we have used meta typing in order to distinguish ordinary and aggregation edges representing refinement in hierarchical graphs. We have pointed out that, due to the notion of morphism between graphs with two levels of typing, the transformation is restricted to the instance level while the type level remains static. In this section, it is our aim to extend the transformation to the type level. To this aim, we provide another construction of graphs with two-level typing which yields the same notion of graphs but a more flexible kind of morphism.

An *arrow category* $\mathbf{C}^{\rightarrow}$ in a given category $\mathbf{C}$ has arrows $a : A \rightarrow A'$ of $\mathbf{C}$ as objects. A morphism between $a : A \rightarrow A'$ and $b : B \rightarrow B'$ is a pair of arrows $\langle f, f' \rangle$ in $\mathbf{C}$ such that the square below on the right commutes.

$$\begin{tikzcd} A \arrow[d, "a"] \\ A' \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "f"] \arrow[d, "a"'] & B \arrow[d, "b"] \\ A' \arrow[r, "f'"] & B' \end{tikzcd}$$

The arrow category $\mathbf{Graph}^{\rightarrow}$ over the category of graphs has graph morphisms as objects. We may think of them as typing morphisms from instance graphs to type graphs. With this intuition, morphisms of $\mathbf{Graph}^{\rightarrow}$ consist of graph morphisms both on the instance and the type level thus allowing the variation of types as required for evolutionary change.

In order to preserve the hierarchical structure established through meta typing and to provide some control on the transformation of types, we build the arrow category over the category of TG_0-typed graphs. Thus, objects of this category are arrows in $\mathbf{Graph}_{TG_0}$ like below on the left, and arrows are pairs of

arrows $\langle f_I, f_T \rangle$ in $\mathbf{Graph}_{TG_0}$ as shown on the right:

$$\begin{tikzcd} G \arrow[r, "g"] \arrow[dr, "g_0"'] & TG \arrow[d, "tg_0"] \\ & TG_0 \end{tikzcd} \qquad \begin{tikzcd} G \arrow[r, "g"] \arrow[rr, bend left, "f_I"] \arrow[drr, bend right, "g_0"'] & TG \arrow[r, "f_T"] \arrow[dr, "tg_0"] & TH \arrow[l, "h"'] \arrow[d, "th_0"] & H \arrow[l] \arrow[dl, bend left, "h_0"] \\ & & TG_0 & \end{tikzcd}$$

Now we can define our ultimate notion of transformation, catering for hierarchical structures as well as for type-level evolution, by replacing the category of TG-typed graphs in Definition 2 with the category $\mathbf{Graph}^{\rightarrow}_{TG_0}$.

Transformation rules in this setting consist of two-level graphs. They modify consistently the type and the instance level. The consistency between the two levels of transformation and the meta typing realizing the hierarchical structure is implicit in the commutativity of the above diagrams.

A more general approach to system evolution has been presented in [38]. In addition to the evolution of type graphs, in [38] also the transformation rules specifying dynamic change may be modified by application of higher-level *evolution rules*. Thus, unlike in our approach, dynamic change and evolution form orthogonal dimensions.

4 Conclusion

In this paper, we have proposed a conceptual framework for mobile and distributed applications. Three orthogonal dimensions of such systems have been identified: A multi-layered, non-strict *hierarchical graph structure*, the *typing* of instance graphs over type graphs, and the *dynamic and evolutionary change* occurring, respectively, on the instance and the type level. This conceptual framework is formalized using concepts of graph transformation, in particular, typed graph transformation systems according to the categorical double-pushout (DPO) approach [18,9] and hierarchical graphs. The use of categorical constructions like comma categories allows us to express the orthogonality and consistency of the three dimensions while inheriting many theoretical results.

Our framework does not yet cover all important aspects of software development. Some of these aspects have been investigated in the graph transformation literature for simpler kinds of systems (e.g., without typing or hierarchical structure). In particular, the software development process with its different phases of analysis, design, and implementation and the refinement steps between these phases are subject of ongoing research (see, e.g., [29,25,37,53]). Horizontal structuring techniques for models and programs like modules, views, or packages have been studied, for example, in [31,30,32,49]. In order to support the transition from design to implementation, high-level programming and modeling languages based on graph transformation are developed (see, e.g., [23,50,22]).

Most of these approaches are neither consistent nor strictly orthogonal to the conceptual framework presented her. It remains future work to identify the

relation between the various concepts and to incorporate them into a consistent overall model of software development.

Acknowledgment

The authors are grateful to Stefan Sauer for fruitful comments on earlier versions of this article.

References

1. M. Andries, G. Engels, A. Habel, B. Hoffmann, H.-J. Kreowski, S. Kuske, D. Plump, A. Schürr, and G. Taentzer. Graph transformation for specification and programming. *Science of Computer Programming*, 34:1–54, 1999.
2. P. Baldan, A. Corradini, H. Ehrig, M. Löwe, U. Montanari, and F. Rossi. Concurrent semantics of algebraic graph transformation. In Ehrig et al. [16].
3. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1–3):109 – 137, 1984.
4. G. Berry and G. Boudol. The chemical abstract machine. *Theoret. Comput. Sci.*, 96(1):217–248, 1992.
5. M.R. Berthold, I. Fischer, and Koch. M. Attributed graph transformation with partial attribution. In H. Ehrig and G. Taentzer, editors, *Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems (GraTra2000)*. Tech. Univ. of Berlin, March 2000. Tech. Report 2000-2, Dept. of Comp. Sci.
6. G. Busatto, G. Engels, K. Mehner, and A. Wagner. A framework for adding packages to graph transformation approaches. In Ehrig et al. [15].
7. Microsoft Corp. Distributed component object model protocol – DCOM, V 1.0, 1998. `http://www.microsoft.com/com/tech/dcom.asp`.
8. A. Corradini, R. Heckel, and U. Montanari. Graphical operational semantics. In *Proc. ICALP2000 Workshop on Graph Transformation and Visual Modelling Techniques, Geneva, Switzerland*. Carleton Scientific, July 2000.
9. A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3,4):241–266, 1996.
10. A. Corradini, U. Montanari, F. Rossi, H. Ehrig, R. Heckel, and M. Löwe. Algebraic approaches to graph transformation, Part I: Basic concepts and double pushout approach. In Rozenberg [47].
11. B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In Rozenberg [47].
12. F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. In J. Tiuryn, editor, *Foundations of Software Science and Computation Structures (FoSSACS'00), Berlin, Germany*, volume 1784 of *LNCS*. Springer Verlag, March/April 2000.
13. F. Drewes, H.-J. Kreowski, and A. Habel. Hyperedge replacement graph grammars. In Rozenberg [47].
14. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 2: Applications, Languages, and Tools*. World Scientific, 1999.
15. H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98), Paderborn, November 1998*, volume 1764 of *LNCS*. Springer Verlag, 2000.

16. H. Ehrig, H.-J. Kreowski, U. Montanari, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 3: Concurrency and Distribution.* World Scientific, 1999.
17. H. Ehrig and M. Löwe. Categorical principles, techniques and results for high-level replacement systems in computer science. *Applied Categorical Structures*, 1(1):21–50, 1993.
18. H. Ehrig, M. Pfender, and H.J. Schneider. Graph grammars: an algebraic approach. In *14th Annual IEEE Symposium on Switching and Automata Theory*, pages 167–180. IEEE, 1973.
19. G. Engels, J.H. Hausmann, R. Heckel, and St. Sauer. Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. Submitted.
20. G. Engels, R. Heckel, and St. Sauer. UML - A universal modeling language? In *Proc. 21st Int. Conference on Application and Theory of Petri Nets (Petri Nets 2000), Aarhus, Denmark*, LNCS. Springer Verlag, June 2000. To appear.
21. G. Engels and A. Schürr. Hierarchical graphs, graph types and meta types. *Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS*, 2, 1995. `http://www.elsevier.nl/locate/entcs/volume2.html`.
22. C. Ermel, M. Rudolf, and G. Taentzer. The AGG approach: Language and tool environment. In Engels et al. [14].
23. T. Fischer, J. Niere, L. Torunski, and A. Zündorf. Story diagrams: A new graph transformation language based on UML and Java. In Ehrig et al. [15].
24. C. Ghezzi, M. Jazayeri, and D. Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall Int., 1991.
25. M. Große-Rhode, F. Parisi Presicce, and M. Simeoni. Refinement of graph transformation systems via rule expressions. In Ehrig et al. [15].
26. Object Management Group. Meta object facility (MOF) specification, 1999. `http://www.omg.org`.
27. Object Management Group. OMG Unified Modeling Language Specification, V 1.3, 1999. `http://www.omg.org`.
28. The Object Management Group. The common object request broker: Architecture and specification, v. 3.0, 2000. `http://www.omg.org`.
29. R. Heckel, A. Corradini, H. Ehrig, and M. Löwe. Horizontal and vertical structuring of typed graph transformation systems. *Math. Struc. in Comp. Science*, 6(6):613–648, 1996.
30. R. Heckel, G. Engels, H. Ehrig, and G. Taentzer. Classification and comparison of modularity concepts for graph transformation systems. In Engels et al. [14].
31. R. Heckel, G. Engels, H. Ehrig, and G. Taentzer. A view-based approach to system modelling based on open graph transformation systems. In Engels et al. [14].
32. R. Heckel, B. Hoffmann, P. Knirsch, and S. Kuske. Simple modules for GRACE. In Ehrig et al. [15].
33. R. Heckel and A. Wagner. Ensuring consistency of conditional graph grammars – a constructive approach. *Proc. of SEGRAGRA'95 "Graph Rewriting and Computation", Electronic Notes of TCS*, 2, 1995. `http://www.elsevier.nl/locate/entcs/volume2.html`.
34. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
35. Sun Microsystems Inc. Java card applet developer's guide. `http://java.sun.com/products/javacard21.html`, 1998.
36. Sun Microsystems Inc. Java remote method invocation. `http://java.sun.com/products/jdk/rmi`, 1998.

37. C.-A. Krapp, S. Krüppel, A. Schleicher, and B. Westfechtel. UML packages for PROgrammed Graph REwrite Systems. In Ehrig et al. [15].
38. M. Löwe. Evolution patterns. Postdoctoral thesis, Technical University of Berlin. Tech. Report 98-4, Dept. of Comp. Sci, 1997.
39. M. Löwe, M. Korff, and A. Wagner. An algebraic framework for the transformation of attributed graphs. In M.R. Sleep, M.J. Plasmeijer, and M.C. van Eekelen, editors, *Term Graph Rewriting: Theory and Practice*, chapter 14, pages 185–199. John Wiley & Sons Ltd, 1993.
40. S. Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, New York, 1971.
41. J. Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoret. Comput. Sci.*, 96:73–155, 1992.
42. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
43. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100:1–77, 1992.
44. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, Computer Science Department, 1981.
45. T.W. Pratt. Definition of programming language semantics using grammars for hierarchical graphs. In H. Ehrig, V. Claus, and G. Rozenberg, editors, *1st Int. Workshop on Graph Grammars and their Application to Computer Science and Biology, LNCS 73*, volume 73 of *LNCS*. Springer Verlag, 1979.
46. W. Reisig. *Petri Nets*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985.
47. G. Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation, Volume 1: Foundations*. World Scientific, 1997.
48. H.-J. Schneider. On categorical graph grammars integrating structural transformation and operations on labels. *Theoret. Comput. Sci.*, 109:257 – 274, 1993.
49. A. Schürr and A.J. Winter. UML packages for PROgrammed Graph REwrite Systems. In Ehrig et al. [15].
50. A. Schürr, A.J. Winter, and A. Zündorf. The PROGRES approach: Language and environment. In Engels et al. [14].
51. M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.
52. G. Taentzer, I. Fischer, M. Koch, and V. Volle. Distributed graph transformation with application to visual design of distributed systems. In Ehrig et al. [16].
53. Andreas Zamperoni and Gregor Engels. Formal integration of software engineering aspects using graph rewrite systems - a typical experience?! In A. Schürr M. Nagl, editor, *Proc. Applications of Graph Transformations With Industrial Relevance (AGTIVE), Kerkrade (The Netherlands), September 1–3, 1999*, LNCS. Springer Verlag, 2000. To appear.

Monotone Proofs of the Pigeon Hole Principle

Albert Atserias *,**, Nicola Galesi **, and Ricard Gavaldà **

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Barcelona, Spain
{atserias,galesi,gavalda}@lsi.upc.es

Abstract. We study the complexity of proving the Pigeon Hole Principle (PHP) in a monotone variant of the Gentzen Calculus, also known as Geometric Logic. We show that the standard encoding of the PHP as a monotone sequent admits quasipolynomial-size proofs in this system. This result is a consequence of deriving the basic properties of certain quasipolynomial-size monotone formulas computing the boolean threshold functions. Since it is known that the shortest proofs of the PHP in systems such as Resolution or Bounded Depth Frege are exponentially long, it follows from our result that these systems are exponentially separated from the monotone Gentzen Calculus. We also consider the monotone sequent (CLIQUE) expressing the *clique-coclique* principle defined by Bonet, Pitassi and Raz (1997). We show that monotone proofs for this sequent can be easily reduced to monotone proofs of the one-to-one and onto PHP, and so CLIQUE also has quasipolynomial-size monotone proofs. As a consequence, Cutting Planes with polynomially bounded coefficients is also exponentially separated from the monotone Gentzen Calculus. Finally, a simple simulation argument implies that these results extend to the Intuitionistic Gentzen Calculus. Our results partially answer some questions left open by P. Pudlák.

1 Introduction

One of the main approaches to attack the **NP** $\neq$ co-**NP** question is that of studying the length of proofs in propositional calculi. In a well-known result, Cook and Reckhow [16] proved that if all propositional proof systems are not *polynomially bounded*, that is, if they have families of tautologies whose shortest proofs are superpolynomial in the size of the formulas, then **NP** $\neq$ co-**NP**. In spite of the simplicity of propositional proof systems such as the Hilbert Calculus (Frege system) or the Gentzen sequent Calculus, we are admitedly far at present from proving that these systems are not polynomially bounded. Surprisingly, one

* Supported by the CUR, Generalitat de Catalunya, through grant 1999FI 00532.
** Partially supported by ALCOM-FT, IST-99-14186, by FRESCO PB98-0937-C04-04 and by SGR CIRIT 1997SGR–00366.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 151–162, 2000.
© Springer-Verlag Berlin Heidelberg 2000

of the main difficulties is the lack of families of tautologies candidate to be hard for these systems.

Nevertheless several important results have been obtained for less powerful but not trivial proof systems. Strong lower bounds are actually known for systems such as Resolution [18,13,5,32,14], Bounded Depth Frege [1,4] and Polynomial Calculus [28]. The common point among these results is the family of formulas that is considered to give the exponential lower bounds. These formulas encode a basic combinatorial principle known as the Pigeon Hole Principle (PHP_n^m), saying that there is no one-to-one mapping from a set of m elements to a set of n elements, provided $m > n$. Resolution was the first proof system for which an exponential lower bound was proved for the size of refutations of the PHP_n^{n+1}, a well-known result due to Haken [18]. This result was generalized to PHP_n^m, for m linear in n, by Buss and Turan [13]. The same formula, PHP_n^{n+1}, was later used by Ajtai [1] to give a superpolynomial size lower bound for a system that subsumes Resolution: Bounded Depth Frege. This result was simplified and improved up to an exponential lower bound by Beame et al. [4]. The complexity of the PHP_n^m is also well-studied in algebraic-style propositional proof systems. Recently, Razborov [28] (see also [19]) showed that PHP_n^m is also hard for the Polynomial Calculus (notice that Riis [30] showed that a different encoding of PHP_n^{n+1} restricted to bijective maps has constant degree proofs in the Polynomial Calculus). Actually one of the most interesting problems is to know the exact complexity of Resolution refutations of PHP_n^m, when $m \geq \frac{n^2}{\log n}$ [6,12,29]. Thus, in spite of its simple combinatorial nature, PHP_n^{n+1} is one of the most commonly used principles to give proof complexity lower bounds. For this reason, in studying the complexity of a new proof system, it is important to consider the complexity of proving PHP_n^{n+1} as a first step. After Haken's lower bound, it was conjectured that PHP_n^{n+1} would also be hard to prove for more powerful proof systems, such as Frege. The conjecture was refuted by Buss [9], who exhibited polynomial-size proofs in Frege, or equivalently, in the Gentzen Calculus. It is also known that PHP_n^{n+1} has polynomial-size proofs in Cutting Planes [17], and that the slightly weaker form PHP_n^{2n} has quasipolynomial-size proofs in Bounded Depth Frege [23,22].

Monotone proof systems, that is, proof systems restricted to propositional formulas over the monotone basis $\{\wedge, \vee\}$, were considered by Pudlák and Buss [26], and more recently, by Pudlák [24], and Clote and Setzer [15]. There are several alternative definitions of monotone proof systems. Here we consider the Monotone Gentzen Calculus, called *Geometric Logic* in [24]. Although the only monotone tautological formula is the true constant 1, Pudlák suggests the study of tautological sequents of the form $A \to B$, where A and B are boolean formulas built over the monotone basis $\{\wedge, \vee\}$. Several interesting combinatorial principles can be put in this form; for example, PHP_n^{n+1}.

The correpondence between circuit complexity classes and proof systems inspires new techniques to obtain both upper and lower bounds for proofs. Examples are the lower bound of Beame et. al. [4] for Bounded Depth Frege (also known as $\mathbf{AC}_0$ Frege), in which they used an adaptation of Hastad's Switch-

ing Lemma, and the polynomial upper bound of Buss [10] for PHP_n^m in Frege (or $\mathbf{NC}_1$-Frege) using an $\mathbf{NC}_1$ circuit for addition. While strong lower bounds for monotone circuits were given more than ten years ago (see [27,3]), non-trivial lower bounds for monotone proof systems are not known yet. Hence, one of the basic questions is whether PHP_n^{n+1} can be used to obtain exponential lower bounds for these systems. This question is also important since the (non-monotone) Frege proofs of PHP_n^{n+1} given by Buss [9] formalize a counting argument, and it is not clear how to formalize counting arguments into short monotone proofs. See the paper by Pudlák [24] for a further discussion on this topic (see also [15]).

In this work we exhibit quasipolynomial-size proofs of PHP_n^{n+1} in the Monotone Gentzen Calculus. To obtain this result, we consider quasipolynomial-size monotone formulas to compute the boolean threshold functions. While polynomial-size monotone formulas are known for these functions [33,2], Pudlák remarks that it is not clear whether their basic properties have short monotone proofs. First, Valiant's construction [33] is probabilistic, and therefore, it does not provide any explicit formula to work with. Second, the sorting network of Ajtai, Komloś, and Szemeredi [2] makes use of expanders graphs, and there is little hope that their basic properties will have short monotone proofs. Here we address the difficulty raised by Pudlák by considering explicit quasipolynomial-size monotone formulas $\mathrm{th}_k^n(x_1, \ldots, x_n)$ to compute threshold functions. We show that the basic properties of $\mathrm{th}_k^n(x_1, \ldots, x_n)$ admit quasipolynomial-size monotone proofs. In particular, we prove that for any permutation π the sequent $\mathrm{th}_k^n(x_1, \ldots, x_n) \vdash \mathrm{th}_k^n(x_{\pi(1)}, \ldots, x_{\pi(n)})$ has quasipolynomial-size monotone proofs.

We remark that our proofs can be made tree-like, but details are omitted in this version. For non-monotone Gentzen Calculi, Krajíček [20] proved that tree-like proofs are as powerful as the unrestricted ones. But it is not known at present whether this holds for the monotone case, as the same technique does not apply.

We also consider the formula CLIQUE_k^n expressing the (n, k)-Clique-Coclique Principle, used by Bonet, Pitassi and Raz, and for which an exponentialy lower bound in Cutting Planes with polynomially bounded coefficients (poly-CP) was proved [8] (notice the difference with the Clique Principle with common variables introduced by Krajíček in [21], and used by Pudlák in [25] to obtain exponential lower bounds for Cutting Planes with unrestricted coefficients. The latter is not a monotone tautology of the form $A \to B$). We show that monotone proofs for the monotone sequent obtained from the formula CLIQUE_k^n can be reduced to monotone proofs of the *onto* version of PHP_{k-1}^k, which in turn can be easily reduced to the standard PHP_{k-1}^k. This way, we obtain quasipolynomial-size monotone proofs of CLIQUE_k^n

Our results imply that Resolution, Bounded-depth Frege, and poly-CP are exponentially separated from the (tree-like) Monotone Gentzen Calculus. Finally, as remarked in [24], a simple simulation argument shows that every proof in

the Monotone Gentzen Calculus, is also a proof in the Intuitionistic Gentzen Calculus. Hence, all our results also hold for this system.

2 Preliminaries

A *monotone formula* is a propositional formula without negations. The *Monotone Gentzen Calculus* (MLK), also called *Geometric Logic* [24], is obtained from the standard Gentzen Calculus (LK [31]) when only monotone formulas are considered, and the negation rules are ignored. As usual, a proof in MLK is a sequence of *sequents*, or lines, of the form $\Gamma \vdash \Delta$ each of which is either an initial axiom, or has been obtained by a rule of MLK from two previous lines in the sequence. The sequence constitutes a proof of the last sequent. When we restrict the proofs in such a way that each derived sequent can be used only once as a premise in a rule of the proof, we say that the system is tree-like.

The overall number of *symbols* used in a proof is the *size* of the proof. Let A and $B_1, \ldots, B_n$ be formulas, and let $x_1, \ldots, x_n$ be propositional variables that may or may not occur in A. We let $A(x_1/B_1, \ldots, x_n/B_n)$ denote the formula that results from A when all occurrences of x_i (if any) are replaced by B_i (replacements are made simultaneously). Observe that if A and B are monotone formulas, then $A(x/B)$ is also monotone. The non-monotone version of the following Lemma appears in [7,10] (monotonicity is only needed in part (v), and the proof is straightforward).

Lemma 1. *For every monotone formula A, the sequents (i) $A, x \vdash A(x/1)$, (ii) $A \vdash x, A(x/0)$, (iii) $A(x/1), x \vdash A$, (iv) $A(x/0) \vdash x, A$, and (v) $A(x/0) \vdash A(x/1)$, have* MLK*-proofs of size quadratic in the size of A.*

For every n and $k \in \{0, \ldots, n\}$, let $\mathrm{TH}_k^n : \{0,1\}^n \to \{0,1\}$ be the boolean function such that $\mathrm{TH}_k^n(a_1, \ldots, a_n) = 1$ if and only if $\sum_{i=1}^k a_i \geq k$, for every $(a_1, \ldots, a_n) \in \{0,1\}^n$. Each TH_k^n is called a threshold function. Valiant [33] proved that every threshold function TH_k^n is computable by a monotone formula of size polynomial in n. The proof being probabilistic, the construction is not explicit. In the same paper, Valiant mentioned that a divide and conquer strategy leads to explicit quasipolynomial-size monotone formulas for all threshold functions. The same construction appears in the book by Wegener [35], and in the more recent book by Vollmer [34]. Here we revisit that construction with a minor modification. We define monotone formulas $\mathrm{th}_0^1(x) := 1$ and $\mathrm{th}_1^1(x) := x$, and for every $n > 1$ and $k \in \{0, \ldots, n\}$, define the formula

$$\mathrm{th}_k^n(x_1, \ldots, x_n) := \bigvee_{(i,j) \in I_k^n} (\mathrm{th}_i^{n/2}(x_1, \ldots, x_{n/2}) \wedge \mathrm{th}_j^{n-n/2}(x_{n/2+1}, \ldots, x_n)),$$

where $I_k^n = \{(i,j) : 0 \leq i \leq n/2,\ 0 \leq j \leq n - n/2,\ i + j \geq k\}$ and $n/2$ is an abbreviation for $\lfloor n/2 \rfloor$. It is straightforward to prove that $\mathrm{th}_k^n(x_1, \ldots, x_n)$ computes the boolean function TH_k^n. On the other hand, it is easy to prove, by induction on n, that the size of $\mathrm{th}_k^n(x_1, \ldots, x_n)$ is bounded by $n^{c \log_2(n)}$ for some constant $c > 0$; that is, the size of $\mathrm{th}_k^n(x_1, \ldots, x_n)$ is quasipolynomial in n.

3 Basic Properties of Threshold Formulas

We establish a number of lemmas stating that the elementary properties of the threshold formulas admit short MLK-proofs. Here, short means size polynomial in the size of the formula $\mathrm{th}_k^n(x_1, \ldots, x_n)$, and therefore, size quasipolynomial in n. The first properties are easy:

Lemma 2. *For every $n, m, k \in \mathbb{N}$ with $m \leq n/2$, and $k \leq n - n/2$, and for every $h, s \in \mathbb{N}$ with $n \geq h \geq s$, the sequents (i) $\vdash \mathrm{th}_0^n(x_1, \ldots, x_n)$, (ii) $\mathrm{th}_n^n(x_1, \ldots, x_n) \vdash \bigwedge_i x_i$, (iii) $\mathrm{th}_m^{n/2}(x_1, \ldots, x_{n/2}) \wedge \mathrm{th}_k^{n-n/2}(x_{n/2+1}, \ldots, x_n) \vdash \mathrm{th}_{m+k}^n(x_1, \ldots, x_n)$, and (iv) $\mathrm{th}_h^n(x_1, \ldots, x_n) \vdash \mathrm{th}_s^n(x_1, \ldots, x_n)$ have* MLK*-proofs of size quasipolynomial in n.*

In the next lemmas we give MLK-proofs of the basic properties relative to the symmetry of the threshold formulas (Theorem 1 below).

Lemma 3. *For every $n, m, k, l \in \mathbb{N}$, with $0 < m \leq n$, $0 \leq k < n$, and $0 \leq l \leq n$, the sequents*

(i) $\mathrm{th}_{k+1}^n(x_1, \ldots, x_l/1, \ldots, x_n) \vdash \mathrm{th}_k^n(x_1, \ldots, x_l/0, \ldots, x_n)$
(ii) $\mathrm{th}_{m-1}^n(x_1, \ldots, x_l/0, \ldots, x_n) \vdash \mathrm{th}_m^n(x_1, \ldots, x_l/1, \ldots, x_n)$

have MLK*-proofs of size quasipolynomial in n.*

Proof: We first show (i). We use induction on n, where the base case is $\mathrm{th}_1^1(1) \vdash \mathrm{th}_0^1(0)$. Assume without loss of generality that $l \leq n/2$, that is, x_l is in the first half of the variables. Recall the definition of $\mathrm{th}_{k+1}^n(x_1, \ldots, x_l/1, \ldots, x_n)$:

$$\bigvee_{(i,j) \in I_{k+1}^n} (\mathrm{th}_i^{n/2}(x_1, \ldots, x_l/1, \ldots, x_{n/2}) \wedge \mathrm{th}_j^{n-n/2}(x_{n/2+1}, \ldots, x_n)).$$

Fix $(i, j) \in I_{k+1}^n$, let $p = n/2$, and let $q = n - n/2$. If $i = 0$, then $j \geq k + 1$ and $\mathrm{th}_j^q(x_{n/2+1}, \ldots, x_n) \vdash \mathrm{th}_k^q(x_{n/2+1}, \ldots, x_n)$ by part (iv) of Lemma 2. Since $\vdash \mathrm{th}_0^p(x_1, \ldots, x_l/0, \ldots, x_{n/2})$ by part (i) of Lemma 2, right $\wedge$-introduction gives $\mathrm{th}_j^q(x_{n/2+1}, \ldots, x_n) \vdash \mathrm{th}_0^p(x_1, \ldots, x_l/0, \ldots, x_{n/2}) \wedge \mathrm{th}_k^q(x_{n/2+1}, \ldots, x_n)$, and so $\mathrm{th}_j^q(x_{n/2+1}, \ldots, x_n) \vdash \mathrm{th}_k^n(x_1, \ldots, x_l/0, \ldots, x_n)$ by a cut with part (iii) of Lemma 2. Left weakening and left $\wedge$-introduction gives then $\mathrm{th}_i^p(x_1, \ldots, x_l/1, \ldots, x_{n/2}) \wedge \mathrm{th}_j^q(x_{n/2+1}, \ldots, x_n) \vdash \mathrm{th}_k^n(x_1, \ldots, x_l/0, \ldots, x_n)$ as desired. If $i > 0$, we have $\mathrm{th}_i^p(x_1, \ldots, x_l/1, \ldots, x_{n/2}) \vdash \mathrm{th}_{i-1}^p(x_1, \ldots, x_l/0, \ldots, x_{n/2})$ by induction hypothesis on n. Therefore, easy manipulation using part (iii) of Lemma 2 as before gives $\mathrm{th}_i^p(x_1, \ldots, x_l/1, \ldots, x_{n/2}) \wedge \mathrm{th}_j^q(x_{n/2+1}, \ldots, x_n) \vdash \mathrm{th}_{i-1+j}^n(x_1, \ldots, x_l/0, \ldots, x_n)$. Finally, since $i - 1 + k \geq k$, a cut with part (iv) of Lemma 2 gives the result. The proof of (ii) is very similar. □

Lemma 4. *For every $m, n, k, l \in \mathbb{N}$ with $1 \leq k < l \leq n$, and $m \leq n$, the sequents*

(i) $\mathrm{th}_m^n(x_1, \ldots, x_k/1, \ldots, x_l/0, \ldots, x_n) \vdash \mathrm{th}_m^n(x_1, \ldots, x_k/0, \ldots, x_l/1, \ldots, x_n)$

(ii) $\mathrm{th}^n_m(x_1,\ldots,x_k/0,\ldots,x_l/1,\ldots,x_n) \vdash \mathrm{th}^n_m(x_1,\ldots,x_k/1,\ldots,x_l/0,\ldots,x_n)$

have MLK-*proofs of size quasipolynomial in* n.

Proof: Both proofs are identical. It is enough to prove (i) when $k \leq n/2 < l$, that is, when x_k falls in the first half of the variables and x_l falls in the second half of the variables. The complete proof of (i) would then be a simple induction on the recursive definition of $\mathrm{th}^n_m(x_1,\ldots,x_k/1,\ldots,x_l/0,\ldots,x_n)$ whose base case is when $k \leq n/2 < l$. Notice that the base case is eventually reached, at latest, when $n = 2$. So assume $k \leq n/2 < l$ and recall the definition of $\mathrm{th}^n_m(x_1,\ldots,x_k/1,\ldots,x_l/0,\ldots,x_n)$:

$$\bigvee_{(i,j)\in I^n_m} (\mathrm{th}^{n/2}_i(x_1,\ldots,x_k/1,\ldots,x_{n/2}) \wedge \mathrm{th}^{n-n/2}_j(x_{n/2+1},\ldots,x_l/0,\ldots,x_n)).$$

Fix $(i,j) \in I^n_m$, let $p = n/2$, and let $q = n - n/2$. If $i > 0$, then Lemma 3 shows that $\mathrm{th}^p_i(x_1,\ldots,x_k/1,\ldots,x_{n/2}) \vdash \mathrm{th}^p_{i-1}(x_1,\ldots,x_k/0,\ldots,x_{n/2})$. Similarly, $\mathrm{th}^q_j(x_{n/2+1},\ldots,x_l/0,\ldots,x_n) \vdash \mathrm{th}^q_{j+1}(x_{n/2+1},\ldots,x_l/1,\ldots,x_n)$ whenever $j < n - n/2$. From these two sequents, the result follows easily when $i > 0$ and $j < n - n/2$. Consider next the case in which either $i = 0$ or $j = n - n/2$. If $j = n - n/2$, then $\mathrm{th}^q_{n-n/2}(x_{n/2+1},\ldots,x_l/0,\ldots,x_n)$ is just provably false by part (ii) of Lemma 2, and the result follows easily. If $i = 0$, then $\mathrm{th}^p_i(x_1,\ldots,x_k/0,\ldots,x_{n/2})$ is just provably true by part (i) of Lemma 2. On the other hand, $\mathrm{th}^q_j(x_{n/2+1},\ldots,x_l/0,\ldots,x_n) \vdash \mathrm{th}^q_j(x_{n/2+1},\ldots,x_l/1,\ldots,x_n)$ follows by part (v) of Lemma 1, and the result follows too. □

Lemma 5. *For every* $m,n,i,j \in \mathbb{N}$, *with* $m \leq n$ *and* $1 \leq i < j \leq n$, *the sequent* $\mathrm{th}^n_m(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n) \vdash \mathrm{th}^n_m(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)$ *has* MLK-*proofs of size quasipolynomial in* n.

Proof: We split the property according to the four possible truth values of x_i and x_j. Namely, we will give proofs of the following four sequents from which the lemma is immediately obtained by the cut rule.

(i) $\mathrm{th}^n_m(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n), x_i, x_j \vdash \mathrm{th}^n_m(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)$,
(ii) $\mathrm{th}^n_m(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n), x_i \vdash x_j, \mathrm{th}^n_m(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)$,
(iii) $\mathrm{th}^n_m(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n), x_j \vdash x_i, \mathrm{th}^n_m(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)$,
(iv) $\mathrm{th}^n_m(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n) \vdash x_i, x_j, \mathrm{th}^n_m(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)$.

We only show (ii), the rest are similar. Two applications of Lemma 1 give $\mathrm{th}^n_m(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n), x_i \vdash x_j, \mathrm{th}^n_m(x_1,\ldots,1,\ldots,0,\ldots,x_n)$. Lemma 4 gives $\mathrm{th}^n_m(x_1,\ldots,x_i,\ldots,x_j,\ldots,x_n), x_i \vdash x_j, \mathrm{th}^n_m(x_1,\ldots,0,\ldots,1,\ldots,x_n)$, and two more applications of Lemma 1 again give $\mathrm{th}^n_m(x_1,\ldots,0,\ldots,1,\ldots,x_n), x_i \vdash x_j, \mathrm{th}^n_m(x_1,\ldots,x_j,\ldots,x_i,\ldots,x_n)$. Finally, a cut between the last two sequents gives (ii). The size of the proof is quasipolynomial since we are applying Theorem 1 on $\mathrm{th}^n_m()$ whose size is quasipolynomial in n. □

Since every permutation on $\{1,\ldots,n\}$ can be obtained as the composition of (polynomially many) permutations in which only two elements are permuted (transpositions), Lemma 5 easily implies the following theorem.

Theorem 1. *For every $m, n \in \mathbb{N}$, with $m \leq n$, and for every permutation π over $\{1, \ldots, n\}$ the sequent $\mathrm{th}_m^n(x_1, \ldots, x_n) \vdash \mathrm{th}_m^n(x_{\pi(1)}, \ldots, x_{\pi(n)})$ has* MLK-*proofs of size quasipolynomial in n.*

The next two properties state that the smallest threshold formulas are provably equivalent to their usual formulas. The proof is omitted.

Lemma 6. *For every $n \in \mathbb{N}$, the sequents*

(i) $\bigvee_i x_i \dashv\vdash \mathrm{th}_1^n(x_1, \ldots, x_n)$;
(ii) $\bigvee_{i \neq j}(x_i \wedge x_j) \dashv\vdash \mathrm{th}_2^n(x_1, \ldots, x_n)$;

have MLK-*proofs of size polynomial in n.*

The next lemma states that threshold functions split by cases:

Lemma 7. *For every $m, n \in \mathbb{N}$ with m even, $m \leq n$, and n an exact power of two, the sequents*

(i) $\mathrm{th}_{m+1}^n(x_1, \ldots, x_n) \vdash \mathrm{th}_{m/2+1}^{n/2}(x_1, \ldots, x_{n/2}), \mathrm{th}_{m/2+1}^{n/2}(x_{n/2+1}, \ldots, x_n)$,
(ii) $\mathrm{th}_m^n(x_1, \ldots, x_n) \vdash \mathrm{th}_{m/2+1}^{n/2}(x_1, \ldots, x_{n/2}), \mathrm{th}_{m/2}^{n/2}(x_{n/2+1}, \ldots, x_n)$,

have MLK-*proofs of size quasipolynomial in n.*

Proof: We first prove (i). Fix $i, j \leq n/2$ such that $i + j \geq m + 1$. Since m is even, either $i \geq m/2 + 1$ or $j \geq m/2 + 1$ for otherwise $i + j \leq m$. In the former case we get $\mathrm{th}_i^{n/2}(x_1, \ldots, x_{n/2}) \vdash \mathrm{th}_{m/2+1}^{n/2}(x_1, \ldots, x_{n/2}), \mathrm{th}_{m/2+1}^{n/2}(x_{n/2+1}, \ldots, x_n)$ by part (iv) of Lemma 2 and the rule of right weakening. In the latter case we get $\mathrm{th}_j^{n/2}(x_{n/2+1}, \ldots, x_n) \vdash \mathrm{th}_{m/2+1}^{n/2}(x_1, \ldots, x_{n/2}), \mathrm{th}_{m/2+1}^{n/2}(x_{n/2+1}, \ldots, x_n)$, and so the rule of left $\wedge$-introduction puts these together in a single sequent. Since this happens for every $i, j \leq n/2$ such that $i + j \geq m + 1$, we get $\mathrm{th}_{m+1}^n(x_1, \ldots, x_n) \vdash \mathrm{th}_{m/2+1}^{n/2}(x_1, \ldots, x_{n/2}), \mathrm{th}_{m/2+1}^{n/2}(x_{n/2+1}, \ldots, x_n)$ as required. The proof of (ii) is extremely similar. Given $i, j \leq n/2$ such that $i + j \geq m$, either $i \geq m/2 + 1$ or $j \geq m/2$. In the former case, as before using part (iv) of Lemma 2, we have $\mathrm{th}_i^{n/2}(x_1, \ldots, x_{n/2}) \vdash \mathrm{th}_{m/2+1}^{n/2}(x_1, \ldots, x_{n/2}), \mathrm{th}_{m/2}^{n/2}(x_{n/2+1}, \ldots, x_n)$. In the latter case prove $\mathrm{th}_j^{n/2}(x_{n/2+1}, \ldots, x_n) \vdash \mathrm{th}_{m/2+1}^{n/2}(x_1, \ldots, x_{n/2}), \mathrm{th}_{m/2}^{n/2}(x_{n/2+1}, \ldots, x_n)$ as before. Manipulation as in part (i) gives property (ii). $\square$

4 Monotone Proofs of PHP

The *Pigeon Hole Principle* states that if $n + 1$ pigeons go into n holes, then there is some hole with more than one pigeon sitting in it. It is encoded by the following (non-monotone) formula

$$\mathrm{PHP}_n^{n+1} := \bigwedge_{i=1}^{n+1} \bigvee_{j=1}^{n} p_{i,j} \to \bigvee_{k=1}^{n} \bigvee_{\substack{i,j=1 \\ i \neq j}}^{n+1} (p_{i,k} \wedge p_{j,k}).$$

Observe that the Pigeon Hole Principle can be obtained as a monotone sequent simply replacing the symbol $\rightarrow$ above by the symbol $\vdash$. From now on we refer to the left part of the sequent as LPHP_n, and to the right part of the sequent as RPHP_n. The sequent itself is denoted PHP_n.

We first see that PHP_n can be reduced to the case in which n is an exact power of two. The proof of this lemma is easy and omitted.

Lemma 8. *There exists a polynomial $p(n)$ such that, for every $m, S \in \mathbb{N}$, if the sequent PHP_m has a* MLK*-proof of size at most S, then, for every $n \leq m$, the sequent PHP_n has a* MLK*-proof of size at most $S + p(n)$.*

Theorem 2. *The sequents PHP_n have* MLK*-proofs of quasipolynomial-size.*

Proof: We first outline the idea of the proof. From the antecedent of PHP_n we immediately derive that for each pigeon i there is at least one variable $p_{i,j}$ that is true ($\mathrm{th}_1^{n+1}(p_{i,1}, \ldots, p_{i,n})$). We deduce that among all variables grouped by pigeons, at least $n+1$ are true ($\mathrm{th}_{n+1}^{n(n+1)}(p_{1,1}, \ldots, p_{1,n}, \ldots, p_{n+1,1}, \ldots, p_{n+1,n})$). The symmetry of the threshold allows us to show that the same holds when the variables are grouped by holes ($\mathrm{th}_{n+1}^{n(n+1)}(p_{1,1}, \ldots, p_{n+1,1}, \ldots, p_{1,n}, \ldots, p_{n+1,n})$). From this, at least one hole contains two pigeons ($\mathrm{th}_2^{n+1}(p_{1,i}, \ldots, p_{n+1,i})$ for some $i \in \{1, \ldots, n\}$), and this implies RPHP_n.

According to Lemma 8, it is enough to give quasipolynomial size proofs of PHP_n when $n+1$ is a power of two, since there is always a power of two between n and $2n$. So let us assume $n = 2^r - 1$ for some $r \in \mathbb{N}$. For technical reasons in the proof we will consider a *squared* form (instead of rectangular form) of PHP_n where we assume the existence of an $(n+1)$-st hole in which no pigeon can go. So, we introduce $n+1$ new symbols $p_{1,n+1}, \ldots, p_{n+1,n+1}$ that will stand for the constant 0. For every $i \in \{1, \ldots, n+1\}$, let $p_i = (p_{i,1}, \ldots, p_{i,n+1})$, and let $q_i = (p_{1,i}, \ldots, p_{n+1,i})$ (hence $q_{n+1} = (0, \ldots, 0)$ is the sequence of $n+1$ zeros). Consider the following four sequents.

$$\mathrm{LPHP}_n \vdash \textstyle\bigwedge_{i=1}^{n+1} \mathrm{th}_1^{n+1}(p_i) \tag{1}$$

$$\textstyle\bigwedge_{i=1}^{n+1} \mathrm{th}_1^{n+1}(p_i) \vdash \mathrm{th}_{n+1}^{(n+1)^2}(p_1, \ldots, p_{n+1}) \tag{2}$$

$$\mathrm{th}_{n+1}^{(n+1)^2}(p_1, \ldots, p_{n+1}) \vdash \mathrm{th}_{n+1}^{(n+1)^2}(q_1, \ldots, q_{n+1}) \tag{3}$$

$$\mathrm{th}_{n+1}^{(n+1)^2}(q_1, \ldots, q_{n+1}) \vdash \mathrm{RPHP}_n \tag{4}$$

In the next lemmas we show how to prove these sequents with quasipolynomial size MLK-proofs. A MLK-proof of $\mathrm{LPHP}_n \vdash \mathrm{RPHP}_n$ of size quasipolynomial in n will follow by three applications of the cut rule. $\square$

Lemma 9. *Sequent (1) has* MLK*-proofs of size polynomial in n.*

Proof: For each $i \in \{1, \ldots, n+1\}$ derive the sequents $\bigvee_{j=1}^{n} p_{i,j} \vdash \bigvee_{j=1}^{n} p_{i,j} \vee 0$ using right weakening and right $\vee$-introduction. Then, n right $\wedge$-introductions and n left $\wedge$-introductions give $\mathrm{LPHP}_n \vdash \bigwedge_{i=1}^{n+1} \mathrm{th}_1^{n+1}(p_i)$ by the definition of LPHP_n and a cut on part (i) of Lemma 6. The size of the whole proof is quadratic in n. $\square$

Lemma 10. *Sequent (2) has* MLK*-proofs of size quasipolynomial in* n.

Proof: Recall that $n+1 = 2^r$. Let $N = (n+1)^2$. The idea of this proof is to successively pack the conjuncts of the antecedent into a unique threshold formula, following a complete binary tree structure of height $\log_2(n+1) = r$. For every $w \in \{0,1\}^r$, let $p^w = p_{\overline{w}}$, where $\overline{w}$ is the position of w in the lexicographical order on $\{0,1\}^r$. Thus, $p^{0^r} = p_1$ and $p^{1^r} = p_{n+1}$. For every $w \in \{0,1\}^{<r}$, let $p^w = (p^{w0}, p^{w1})$. Observe that $p^\lambda = (p_1, \ldots, p_{n+1})$, where λ is the empty word. For each $t \in \{1, \ldots, r\}$, we exhibit a MLK-proof of

$$\bigwedge_{w \in \{0,1\}^t} \mathrm{th}^{N/2^t}_{(n+1)/2^t}(p^w) \vdash \bigwedge_{w \in \{0,1\}^{t-1}} \mathrm{th}^{N/2^{t-1}}_{(n+1)/2^{t-1}}(p^w) \tag{5}$$

of size quasipolynomial in n. Once we have all these proofs, we only have to cut sequentially to obtain the lemma. We prove sequent 5. For a fixed $t \in \{1, \ldots, r\}$ and a fixed $w \in \{0,1\}^{t-1}$, an application of part (iii) of Lemma 2 gives

$$\mathrm{th}^{N/2^t}_{(n+1)/2^t}(p^{w0}) \wedge \mathrm{th}^{N/2^t}_{(n+1)/2^t}(p^{w1}) \vdash \mathrm{th}^{N/2^{t-1}}_{(n+1)/2^{t-1}}(p^w).$$

We put all these formulas in a unique conjunction using $\wedge$-introduction to get sequent 5. The size of the proof is clearly quasipolynomial in n. □

Lemma 11. *Sequent (3) has* MLK*-proofs of size quasipolynomial in* n.

Proof: Immediate from Theorem 1 because $q_1, \ldots, q_{n+1}$ is a permutation of $p_1, \ldots, p_{n+1}$. □

Lemma 12. *Sequent (4) has* MLK*-proofs of size quasipolynomial in* n.

Proof: The idea of this proof is to unfold the threshold formula in the antecedent into disjunctions of threshold formulas computing the number of pigeons going into each hole. The unpacking process follows the structure of a complete binary tree of height $\log_2(n+1) = r$ in reverse order of that of Lemma 10. We use properties (i) and (ii) of Lemma 7 to perform this process.

Recall that $n+1 = 2^r$. Let $N = (n+1)^2$. Define $q^w = q_{\overline{w}}$ for every $w \in \{0,1\}^r$, where $\overline{w}$ is defined as in the proof of Lemma 10. For every $w \in \{0,1\}^{<r}$, define $q^w = (q^{w0}, q^{w1})$. Observe that $q^\lambda = (q_1, \ldots, q_{n+1})$. For every $t \in \{0, \ldots, r-1\}$ and $w \in \{0,1\}^t$, properties (ii) and (i) of Lemma 7 give

$$\mathrm{th}^{N/2^t}_{(n+1)/2^t}(q^w) \vdash \mathrm{th}^{N/2^{t+1}}_{(n+1)/2^{t+1}+1}(q^{w0}),\ \mathrm{th}^{N/2^{t+1}}_{(n+1)/2^{t+1}}(q^{w1})$$
$$\mathrm{th}^{N/2^t}_{(n+1)/2^t+1}(q^w) \vdash \mathrm{th}^{N/2^{t+1}}_{(n+1)/2^{t+1}+1}(q^{w0}),\ \mathrm{th}^{N/2^{t+1}}_{(n+1)/2^{t+1}+1}(q^{w1}).$$

Appropriate cuts and the definition of q^w for $w \in \{0,1\}^r$ show then that

$$\mathrm{th}^N_{n+1}(q^\lambda) \vdash \mathrm{th}^{n+1}_2(q_1), \mathrm{th}^{n+1}_2(q_2), \ldots, \mathrm{th}^{n+1}_2(q_n), \mathrm{th}^{n+1}_1(q_{n+1}).$$

Since $q_{n+1} = (0, \ldots, 0)$, we immediately have that $\mathrm{th}^{n+1}_1(q_{n+1}) \vdash 0$ by part (ii) of Lemma 6, so that the result follows by a cut on $0 \vdash$, successive cuts on part (iv) of Lemma 6, and right $\vee$-introduction. The size of the proof is again quasipolynomial in n. □

5 Separation Results

A graph G is k-clique if there is a set of k nodes of G such that any two distinct nodes of the set are connected by an edge, and no other edge is present in G. A graph G is a k-coclique if there is a partition of the nodes of G into k disjoint sets in such a way that any two nodes that belong to different sets are connected by an edge, and no other edges are present in G.

The (n,k)-clique-coclique principle of [8] says that, given a set V of n nodes, if G is a k-clique over V and H is a $(k-1)$-coclique over V, then there is an edge in G that is not present in H. This principle may be stated as a monotone sequent CLIQUE_k^n as follows. For every $l \in \{1,\dots,k\}$ and $i \in \{1,\dots,n\}$, let x_{li} be a propositional variable whose intended meaning is that i is the l-th largest node of the fully connected set which forms a fixed k-clique over $\{1,\dots,n\}$. Similarly, for every $l \in \{1,\dots,k-1\}$ and $i \in \{1,\dots,n\}$, let y_{li} be a propositional variable whose intended meaning is that the i-th node is in the l-th disjoint set of a fixed $(k-1)$-coclique over $\{1,\dots,n\}$. The principle is then expressed as follows

$$\bigwedge_{l=1}^{k}\bigvee_{i=1}^{n} x_{li} \wedge \bigwedge_{i=1}^{n}\bigvee_{l'=1}^{k-1} y_{l'i} \vdash \bigvee_{t=1}^{k-1}\bigvee_{\substack{l,l'=1\\ l\neq l'}}^{k}\bigvee_{\substack{i,j=1\\ i\neq j}}^{n}(x_{li}\wedge x_{l'j}\wedge y_{ti}\wedge y_{tj}) \vee \bigvee_{\substack{l,l'=1\\ l\neq l'}}^{k}\bigvee_{i=1}^{n}(x_{li}\wedge x_{l'i}).$$

As in [8], the reduction of CLIQUE_k^n to PHP_{k-1} is accomplished by the substitution of variable $p_{l,l'}$ in PHP_{k-1} by the monotone formula $\bigvee_{i=1}^{n}(x_{li}\wedge y_{l'i})$. The details of the reduction are easy to work out in MLK, and are left to the long version of the paper.

Corollary 1. *The sequents* CLIQUE_k^n *have* MLK*-proofs of quasipolynomial-size.*

Putting together our upper bounds for PHP_n^{n+1} and for CLIQUE_k^n with the exponential lower bounds in Resolution [18], Bounded Depth Frege [1,4], and poly-CP [8], we obtain the following separations result:

Theorem 3. *Resolution, Bounded-Depth Frege and poly-CP are exponentially separated from the Monotone Gentzen Calculus.*

The Intuitionistic Gentzen Calculus forbids sequents with more than one formula in their consequent (see [31] for a precise definition). As observed by Pudlák [24], there is a simple simulation of the Monotone Gentzen Calculus by the Intuitionistic Gentzen Calculus. The simulation consists in replacing consequents with more than one formula by the disjunction of these formulas. This simple simulation implies that all our results also hold for the Intuitionistic Gentzen Calculus.

In [24], Pudlák proves that the Intuitionistic Gentzen Calculus enjoys a feasible interpolation property. It is also asked in [24] whether the feasible interpolation can be made monotone. While we have been able to provide a quasipolynomial upper bound for the size of intuitionisitic proofs of an encoding of the

Clique-Coclique Principle, it is not clear whether the encoding of the Clique Principle on which to apply the interpolation property (the one with common variables as in [21]) enjoys the same upper bound. The reason is that the resulting sequent is not monotone anymore, and our reduction method does not apply. On the other hand, a positive answer would imply that the disjointness property for the Intuitionistic Gentzen Calculus would belong to $\mathbf{P/poly} - \mathbf{mP/poly}$. In fact, the disjointness property would be computable by a (uniform) polynomial-size circuit (see [11] for a proof of this fact), but would not be computable by a monotone polynomial-size circuit, since otherwise, the Intuitionistc Gentzen Calculus would admit the monotone feasible interpolation property.

Acknowledgments. We thank Maria L. Bonet for helpful comments and insights, Pavel Pudlák for comments on a preliminary version and pointing out that our proofs also holds for the tree-like case. Toni Pitassi has informed us that she obtained Theorem 2 independently. We thank her for reading a preliminary version of this paper. We also thank the anonymous ICALP referees for useful comments.

References

1. M. Ajtai. The complexity of the pigeonhole principle. *Combinatorica*, 14, pp. 417-433, 1994.
2. M. Ajtai, J. Komlós, E. Szemerédi. An $O(n \log n)$ sorting network. *Combinatorica*, 3(1), pp. 1-19, 1983.
3. N. Alon, R. B. Boppana. The monotone circuit complexity of boolean functions. *Combinatorica*, 7, pp. 1-22, 1987.
4. P. Beame, R. Impagliazzo, J. Krajíček, T. Pitassi, P. Pudlák, A. Woods. Exponential lower bounds for the Pigeon Hole Principle. *STOC'92*, pp.200-220, 1992.
5. P. Beame, T. Pitassi. Propositional Proof Complexity: Past, Present and Future. *Bulletin of the European Association for Theoretical Computer Science*, 65, 1998.
6. P. Beame, T. Pitassi. Simplified and Improved Resolution Lower Bound. *FOCS'96*, pp. 274-282, 1996.
7. M. Bonet, C. Domingo, R. Gavaldà, A. Maciel, T. Pitassi. Non-automatizability of Bounded-Depth Frege Proofs. *IEEE Conference on Computational Complexity*, 1998.
8. M. Bonet, T. Pitassi, R. Raz. Lower Bounds for Cutting Planes Proofs with small Coefficients. *Journal of Symbolic Logic*, 62 (3), pp. 708-728, 1997. A preliminary version appeared in STOC'95.
9. S. R. Buss. Polynomial size proofs of the propositional pigeon hole principle. *Journal of Symbolic Logic*, 52 (4), pp. 916-927, 1987.
10. S. Buss. Some remarks on length of proofs. *Archive for Mathematical Logic*, 34, pp. 377-394, 1995.
11. S. Buss, G. Mints. The complexity of disjunction and existence properties in intuitionistc logic. Preprint, 1998.
12. S. Buss, T. Pitassi. Resolution and the weak Pigeonhole principle. *Invited Talk to CSL 97. To appear in Selected Papers of the 11-th CSL*, Lecture Notes in Computer Science, 1998.
13. S. R. Buss, G. Turan. Resolution proofs of generalized pigeonhole principles. *Theoretical Computer Science*, 62 (3), pp. 311-317, 1988.

14. V. Chvátal E. Szemerédi. Many hard examples for resolution. *Journal of the Association for Computer Machinery*, 35, pp. 759-768, 1988.
15. P. Clote, A. Setzer. On PHP, st-connectivity and odd charged graphs. *Proof Complexity and Feasible Arithmetics*, 93-118, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol 39, eds. Paul W. Beame and Samuel R. Buss, 1998.
16. S. Cook, R. Reckhow. The relative efficiency of propositional proof systems. *Journal of Symbolic Logic*, 44, pp. 36-50, 1979.
17. W. Cook, C. R. Coullard, G. Turán. On the complexity of Cutting Plane proofs. *Discrete Applied Mathematics*, 18, pp. 25-38, 1987.
18. A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39 (2-3), pp. 297-305, 1985.
19. R. Impagliazzo, P. Pudlak, J. Sgall. Lower Bounds for the Polynomial Calculus and the Groebner basis Algorithm. ECCC TR97-042. To appear in *Computational Complexity.*
20. J. Krajíček. Speed-up for propositional Frege systems via generalizations of proofs, *Commentationes Mathematicae Universiatatis Carolinae*,30, 1989, pp. 137-140.
21. J. Krajíček. Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic. *Journal of Symbolic Logic*, 62, pp. 457-486, 1997.
22. A. Maciel, T. Pitassi, and A. R. Woods. A New Proof of the Weak Pigeonhole Principle. To appear in STOC'00.
23. J. B. Paris, A. J. Wilkie, and A. R. Woods. Provability of the pigeonhole principle and the existence of infinitely many primes. *Journal of Symbolic Logic*, 53 (4), pp. 1235-1244, 1988.
24. P. Pudlák. On the complexity of the propositional Calculus. *Logic Colloquium '97.* To appear.
25. P. Pudlák. Lower bounds for resolutions and cutting planes proofs and monotone computations. *Journal of Symbolic Logic*, 62 (2), pp. , 1997.
26. P. Pudlák, S. Buss. How to lie without being (easily) convicted and the lengths of proofs in propositional calculus. *8th Workshop on CSL, Kazimierz, Poland, September 1994*, Springer Verlag LNCS n.995, pp. 151-162, 1995.
27. A. Razborov. Lower bounds for the monotone complexity of some boolean functions. *Soviet Math. Doklady*, 31 (2), pp. 354-357, 1985.
28. A. Razborov. Lower bounds for the Polynomial Calculus. *Computational Complexity*, 7 (4), pp. 291-324, 1998.
29. A. A. Razborov, A. Wigderson, A. Yao. Read Once Branching Programs, Rectangular Proofs of the Pigeonhole Principle and the Transversal Calculus. *STOC'97*, pp. 739-748, 4-6 May 1997.
30. S. Riis. A Complexity Gap for Tree-Resolution. *BRICS Report Series, RS-99-29*, 1999.
31. G. Takeuti. *Proof Theory.* North-Holland, second edition, 1987.
32. A. Urquhart. Hard examples for Resolution. *Journal of the Association for Computing Machinery*, 34 (1), pp. 209-219, 1987.
33. L. Valiant. Short monotone formulae for the majority function. *Journal of Algorithms*, 5, pp. 363-366, 1984.
34. H. Vollmer. *Introduction to Circuit Complexity.* Springer, 1999.
35. I. Wegener. *The Complexity of Boolean Functions.* J. Wiley and Sons, 1987.

Fully-Abstract Statecharts Semantics via Intuitionistic Kripke Models

Gerald Lüttgen[1] and Michael Mendler[2]

[1] ICASE, Mail Stop 132C, NASA Langley Research Center, Hampton, Virginia 23681-2199, USA, luettgen@icase.edu
[2] Department of Computer Science, Sheffield University, 211 Portobello Street, Sheffield S1 5DP, U.K., M.Mendler@dcs.shef.ac.uk

Abstract. The semantics of Statecharts macro steps, as introduced by Pnueli and Shalev, lacks compositionality. This paper first analyzes the compositionality problem and traces it back to the invalidity of the Law of the Excluded Middle. It then characterizes the semantics via a particular class of linear, intuitionistic Kripke models, namely stabilization sequences. This yields, for the first time in the literature, a simple fully-abstract semantics which interprets Pnueli and Shalev's concept of failure naturally. The results not only give insights into the semantic subtleties of Statecharts, but also provide a basis for developing algebraic theories for macro steps and for comparing different Statecharts variants.

1 Introduction

Statecharts is a well-known design notation for specifying the behavior of embedded systems [6]. It extends *finite state machines* by concepts of *hierarchy* and *concurrency*. Semantically, a Statechart may respond to an event entering the system by engaging in an enabled transition. This may generate new events which, by *causality*, may in turn trigger additional transitions while disabling others. The *synchrony hypothesis* ensures that one execution step, a so-called *macro step*, is complete as soon as this chain reaction comes to a halt.

Pnueli and Shalev presented two equivalent formalizations of Statecharts' macro-step semantics in a seminal paper [16]. However, their semantics violates the desired property of *compositionality*. Huizing and Gerth [10] showed that combining compositionality, causality, and the synchrony hypothesis cannot be done within a simple, single-leveled semantics. Some researchers then devoted their attention to investigating new variants of Statecharts, obeying just two of the three properties. In *Esterel* [3] and *Argos* [15] causality is treated separately from compositionality and synchrony, while in (synchronous) *Statemate* [8] and *UML Statecharts* [7] the synchrony hypothesis is rejected. Other researchers achieved combining all three properties by storing semantic information via preorders [14,17] or transition systems [5,13]. However, no analysis of exactly how much information is needed to achieve compositionality has been made, yet.

This paper first illustrates the compositionality defect of Pnueli and Shalev's semantics by showing that equality of response behavior is not preserved by

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 163–174, 2000.
© Springer-Verlag Berlin Heidelberg 2000

the concurrency and hierarchy operators of Statecharts (cf. Sec. 2). The reason is that macro steps abstract from causal interactions with a system's environment, thereby imposing a closed-world assumption. Indeed, the studied problem can be further traced back to the invalidity of the *Law of the Excluded Middle*. To overcome the problem, we interpret Statecharts, relative to a given system state, as intuitionistic formulas. These are given meaning as specific *intuitionistic Kripke structures* [18], namely linear increasing sequences of event sets, called *stabilization sequences*, which encode interactions between Statecharts and environments. In this domain, which is also characterized algebraically via semilattices, we develop a *fully-abstract* macro-step semantics in two steps. First, we study Statecharts without hierarchy operators. We show that in this fragment, stabilization sequences naturally characterize the largest congruence contained in equality of response behavior (cf. Sec. 3). In the second step, based on a non-standard *distributivity law* and our lattice-theoretic characterization of the intuitionistic semantics, we lift our results to arbitrary Statecharts (cf. Sec. 4). We refer the reader to [12] for the proofs of our results.

2 Statecharts: Notation, Semantics, & Compositionality

Statecharts is a visual language for specifying *reactive systems*, i.e., concurrent systems interacting with their *environment*. They subsume labeled transition systems where labels are pairs of *event* sets. The first component of a pair is referred to as *trigger*, which may include *negated events*, and the second as *action*. Intuitively, a transition is enabled if the environment offers all events in the trigger but not the negated ones. When a transition fires, it produces the events specified in its action. Concurrency is introduced by allowing Statecharts to run in parallel and to communicate by *broadcasting* events. Additionally, *basic states* may be hierarchically refined by injecting other Statecharts.

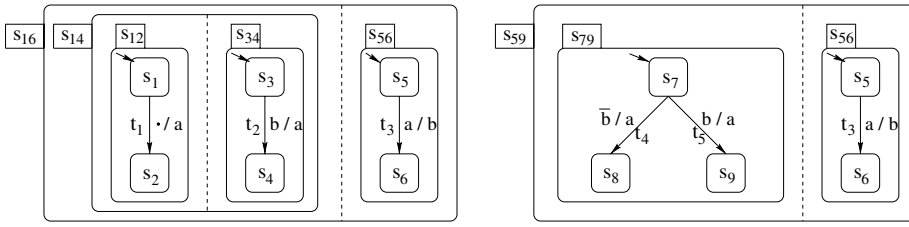


Fig. 1. Two example Statecharts

As an example, the Statechart depicted in Fig. 1 on the left consists of an *and-state* s_{16}, which puts and-state s_{14} and or-state s_{56} in parallel. Similarly, state s_{14} is a parallel composition of or-states s_{12} and s_{34}. Each of these *or-states* describes a sequential state machine and is refined by two *basic states*. In case of s_{12}, basic state s_1 is the initial state which is connected to basic state s_2 via

transition t_1. Here, s_1 is the source state of t_1, state s_2 is its target state, "·" symbolizes its empty trigger, and a is its action. Hence, t_1 is always enabled in the initial state, regardless of the events offered by the environment. Its firing produces event a and switches the active state of s_{12} to s_2. This initiates a causal chain reaction, since the generation of a in turn triggers t_3 which introduces event b. As a consequence, t_2 is enabled and fires within the same *macro step*.

The Statechart depicted in Fig. 1 on the right is like the one on the left, except that and-state s_{14} is replaced by or-state s_{79}. The latter state encodes a choice regarding the execution of t_4 and t_5 from state s_7. The trigger of t_4 is $\bar{b}$, i.e., t_4 is triggered by the absence of event b. Starting with an environment offering no event, thus assuming b to be absent, s_{59} can autonomously engage in t_4. The generation of a in turn triggers t_3, which fires and produces b. However, t_4 was fired under the assumption that b is absent. Since Statecharts is a synchronous language and no event can be both present and absent within a macro step, this behavior is rejected as *globally inconsistent*. Thus, the response of s_{59} to the empty environment is not an empty response but failure.

Statecharts Configurations and Step Semantics. We formalize the Statecharts language relative to a given set of active states. Let Π and $\mathcal{T}$ be countable sets of events and transition names, respectively. For every event $e \in \Pi$, its negated counterpart is denoted by $\overline{e}$. We define $\overline{\overline{e}} =_{\mathrm{df}} e$ and write $\overline{E}$ for $\{\overline{e} \mid e \in E\}$. With every $t \in \mathcal{T}$, we associate a transition $\mathsf{trg}(t)/\mathsf{act}(t)$ consisting of a trigger $\mathsf{trg}(t) \subseteq_{\mathrm{fin}} \Pi \cup \overline{\Pi}$ and an action $\mathsf{act}(t) \subseteq_{\mathrm{fin}} \Pi$, where $\mathsf{trg}(t)$ and $\mathsf{act}(t)$ are required to be finite sets. For simplicity we also write $e_1 \cdots e_n/a_1 \cdots a_m$ for transition $\{e_1, \ldots, e_n\}/\{a_1, \ldots, a_m\}$. The syntax of Statecharts terms is the BNF $C ::= \mathbf{0} \mid x \mid t \mid C \| C \mid C + C$, where $t \in \mathcal{T}$ and x is a variable. Terms not containing variables are called *configurations*. Intuitively, the configuration $\mathbf{0}$ represents a Statechart state with no outgoing transitions (basic state), $C \| D$ denotes the parallel composition of configurations C and D (and-state), and $C + D$ stands for the choice between executing C or D (or-state). The latter construct $+$ coincides with Statecharts' hierarchy operator which reduces to choice on the macro-step level; thus, we refer to operator $+$ also as choice operator. In the standard visual Statecharts notation, $C + D$ is somewhat more restrictive in that it requires D to be a choice of transitions; e.g., $(t_1 \| t_2) + (t_3 \| t_4)$ is prohibited according to Statecharts' syntax, whereas it is a valid configuration in our setting. Semantically, however, our generalization is inessential wrt. the semantics of Pnueli and Shalev which underlies this work (cf. [12]). The set of all configurations is denoted by C and ranged over by C and D. The set of "+"-free, or *parallel*, configurations is written as PC. We call terms $\Phi[x]$ with a single variable occurrence x *contexts* and write $\Phi[C]$ for the substitution of C for x in $\Phi[x]$. Contexts of the form $x \| C$ and $x + C$ are referred to as *parallel contexts* and *choice contexts*, respectively. We tacitly assume that transition names are unique in every term, and we let $\mathsf{trans}(C)$ stand for the set of transition names occurring in C.

Any Statechart in a given set of active states corresponds to a configuration. For example, Statecharts s_{14} and s_{79}, in their initial states (indicated by small arrows in Fig. 1), correspond to $C_{14} =_{\mathrm{df}} t_1 \| t_2$ and $C_{79} =_{\mathrm{df}} t_4 + t_5$, respectively.

The Statecharts depicted in Fig. 1 are then formalized as $C_{16} =_{\mathrm{df}} \Phi_{56}[C_{14}]$ and $C_{59} =_{\mathrm{df}} \Phi_{56}[C_{79}]$, respectively, where $\Phi_{56}[x] =_{\mathrm{df}} x \| t_3$. Moreover, since transitions are uniquely named in configurations and thus may be associated with their source and target states, one can easily determine the set of active states reached after firing a set of transitions; see [16] for details. In this paper, we do not consider *interlevel transitions* and *state references* which would require an extension of our syntax for configurations. However, our semantics should be able to accommodate these features, too.

To present the *response behavior* of a configuration C, as defined by Pnueli and Shalev [16], we have to determine which transitions in $\mathsf{trans}(C)$ may fire together to form a macro step. A macro step comprises a *maximal* set of transitions that are *triggered* by events offered by the environment or produced by the firing of other transitions, that are mutually *consistent* ("orthogonal"), and that obey *causality* and *global consistency*. A transition t is consistent with $T \subseteq \mathsf{trans}(C)$, in signs $t \in \mathsf{consistent}(C, T)$, if t is not in the same parallel component as any $t' \in T$. A transition t is *triggered* by a finite set E of events, in signs $t \in \mathsf{triggered}(C, E)$, if the positive, but not the negative, trigger events of t are in E. Finally, we say that t is *enabled* in C regarding a finite set E of events and a set T of transitions, if $t \in \mathsf{enabled}(C, E, T) =_{\mathrm{df}} \mathsf{consistent}(C, T) \cap \mathsf{triggered}(C, E \cup \bigcup_{t \in T} \mathsf{act}(t))$. Intuitively, assuming transitions T are known to fire, $\mathsf{enabled}(C, E, T)$ determines the set of all transitions of C that are enabled by the actions of T and the environment events in E. We may now present Pnueli and Shalev's *step-construction procedure* for causally determining macro steps:

$\mathsf{procedure}$ *step-construction*$(C,\ E)$; $\mathsf{var}\ T := \emptyset$;
  $\mathsf{while}\ T \subset \mathsf{enabled}(C, E, T)\ \mathsf{do}\ \mathsf{choose}\ t \in \mathsf{enabled}(C, E, T) \setminus T;\ \ T := T \cup \{t\}\ \mathsf{od}$;
  $\mathsf{if}\ T = \mathsf{enabled}(C, E, T)\ \mathsf{then}\ (\mathsf{return}\ T)\ \mathsf{else}\ (\textit{report failure})$

This procedure *nondeterministically* computes, relative to configuration C and finite environment E, those sets T of transitions that can fire together in a macro step. Due to failures raised when detecting global inconsistencies, the construction might involve *backtracking*. The role of failures may be highlighted further by a conservative extension of Pnueli and Shalev's setting that includes an explicit failure event $\bot \in \Pi$. It will be instructive to study the semantics with and without $\bot$ in this paper. Now, for each set T returned by the above procedure, we say that $A =_{\mathrm{df}} E \cup \bigcup_{t \in T} \mathsf{act}(t) \subseteq_{\mathrm{fin}} \Pi$ is a *(step) response*, in signs $C \Downarrow_E A$. When $\bot$ is considered, we also require that $\bot \notin A$. If $E = \emptyset$, we simply write $C \Downarrow A$. Note that E may be modeled by a parallel context consisting of the single transition $\cdot/E$, i.e., $C \Downarrow_E A$ iff $(C \| \cdot/E) \Downarrow A$. This macro-step semantics induces a natural equivalence relation $\sim$ over configurations, called *step equivalence*, satisfying $C \sim D$, whenever $C \Downarrow_E A$ iff $D \Downarrow_E A$, for all $E, A \subseteq_{\mathrm{fin}} \Pi$. For simplicity, $\sim$ does not account for target states of transitions since these can be encoded as event names.

The Compositionality Problem. The compositionality defect of the macro-step semantics manifests itself in the fact that $\sim$ is not a congruence for the configuration algebra. Consider Fig. 1 and assume that states s_2, s_4, s_6, s_8, and s_9 are all equivalent. It is easy to see that configurations C_{14} and C_{79} have

the same response behavior. Both $C_{14} \Downarrow_E A$ and $C_{79} \Downarrow_E A$ are equivalent to $A = E \cup \{a\}$, no matter whether event b is present or absent in environment E. However, $\Phi_{56}[C_{14}] = C_{16} \not\sim C_{59} = \Phi_{56}[C_{79}]$ since $C_{16} \Downarrow \{a, b\}$ but $C_{59} \not\Downarrow A$, for any A. Hence, the equivalence $C_{14} \sim C_{79}$ is not preserved by context $\Phi_{56}[x]$. Intuitively, C_{14} and C_{79} are identified because the response semantics does not account for any interaction with the environment. It adopts the classic *closed-world assumption*, stating that every environment event is either present from the very beginning of a given macro step or will never arise. This eliminates the possibility that events may be generated due to interactions with the environment, in this case event b in $C_{16} \Downarrow \{a, b\}$. In short, a *compositional* macro-step semantics does not validate the *Law of the Excluded Middle* $b \vee \neg b = true$. Since *intuitionistic logic* [18] differs from classic logic by refuting the Law of the Excluded Middle, it is a good candidate framework for analyzing Statecharts semantics. It should be stressed that the compositionality defect is mainly an issue of operator $\|$ and not of $+$, as we will see below.

Our goal is to characterize the largest congruence $\simeq$, called *step congruence*, contained in step equivalence, where $C \simeq D$, if $\Phi[C] \sim \Phi[D]$ for all contexts $\Phi[x]$. Of course, $C \simeq D$ iff $[\![C]\!]_0 = [\![D]\!]_0$, for $[\![C]\!]_0 =_{\mathrm{df}} \{\langle A, \Phi[x]\rangle \mid \Phi[C] \Downarrow A\}$. However, $[\![\cdot]\!]_0$ is a syntactical characterization rather than a semantical characterization which we will develop below. Note that we intend to achieve compositionality in the (declarative) sense of a fully-abstract semantics and not in the (constructive) sense of a denotational semantics.

3 Macro-step Semantics via Stabilization Sequences

We start off by investigating parallel configurations within parallel contexts. We propose a novel semantics for this fragment, show its relation to Pnueli and Shalev's original semantics, and derive a full-abstraction result. Section 4 generalizes this result to arbitrary configurations within arbitrary contexts.

Our new interpretation of parallel configurations C, based on an "open-world assumption," is given in terms of *finite increasing sequences of "worlds"* $E_0 \subset E_1 \subset \cdots \subset E_n$. Each $E_i \subseteq_{\mathrm{fin}} \Pi \setminus \{\bot\}$ is the set of events generated or present in the respective world. The required absence of $\bot$ ensures that each world is consistent. A sequence represents the interactions between C and a potential environment during a macro step. Intuitively, the initial world E_0 contains all events e which are generated by those transitions of C that can fire autonomously. When transitioning from world E_{i-1} to E_i, some events in $E_i \setminus E_{i-1}$ are provided by the environment, as reaction to the events validated by C when reaching E_{i-1}. The new events destabilize world E_{i-1} and may enable a chain reaction of transitions in C. The step-construction procedure, which tracks and accumulates all these events, then defines the new world E_i. Accordingly, we call the above sequences *stabilization sequences*. The overall response of C after n interactions with the environment is the set E_n.

The monotonicity requirement of stabilization sequences reflects the fact that our knowledge of the presence and absence of events increases within the con-

struction of a macro step. Each world contains the events assumed or positively known to be present. Only if an event is not included in the final world, it is known to be absent for sure; the fact that an event e is not present in a world does not preclude e from becoming available later in the considered stabilization sequence. This semantic gap between "not present" and "absent" makes the underlying logic *intuitionistic* as opposed to classic.

Model-Theoretic Semantics for Parallel Configurations. Formally, a stabilization sequence M is a pair (n, V), where $n \in \mathbb{N} \setminus \{0\}$ is its *length* and V is a *state valuation*, i.e., a monotonic mapping from the interval $[0, \ldots, n-1]$ to finite subsets of $\Pi \setminus \{\bot\}$. The *final world* $V(n-1)$ of M is denoted by M^*. We shall assume that M is *irredundant*, i.e. $V(i-1) \subset V(i)$ for all $0 < i < n$, and identify sequences $(1, V)$ of length 1 with subsets $V(0) \subseteq_{\text{fin}} \Pi \setminus \{\bot\}$.

Definition 1. *Let $M = (n, V)$ be a stabilization sequence and $C \in \mathsf{PC}$. Then, M is a* sequence model *of C, written $M \models C$, according to the following clauses: (i) always $M \models \mathbf{0}$; (ii) $M \models C \| D$ iff $M \models C$ and $M \models D$; (iii) $M \models E/A$ iff $\{E \cap \overline{\Pi} \cap V(n-1) = \emptyset$ and $E \cap \Pi \subseteq V(i)\}$ implies $A \subseteq V(i)$, for all $0 \leq i < n$.*

Def. 1 is a shaved version of the standard semantics obtained when reading $C \in \mathsf{PC}$ as an intuitionistic formula [18], i.e., when taking events to be atomic propositions and replacing $\overline{a}$ by negation $\neg a$, concatenation of events and "$\|$" by conjunction "$\wedge$", and "/" by implication "$\supset$". An empty trigger, an empty action, and $\mathbf{0}$ are identified with *true*. Then, $M \models C$ iff C holds for the intuitionistic Kripke structure M. In the sequel we write $SM(C)$ for $\{M \mid M \models C\}$.

In our example $C_{79} = \overline{b}/a + b/a$ is step-congruent to $C'_{79} = \overline{b}/a \,\|\, b/a$ (cf. Sec. 4) which may be identified with formula $(\neg b \supset a) \wedge (b \supset a)$. In classic logic, C'_{79} is equivalent to the single transition $C_{12} = \cdot/a$ corresponding to formula $true \supset a$. As mentioned before, this is inadequate as both have different operational behavior, since $C'_{79} \| \, a/b$ fails in the empty environment whereas $C_{12} \| \, a/b$ has step response $\{a, b\}$. In our intuitionistic semantics, the difference is faithfully witnessed by the stabilization sequence $M = (2, V)$, where $V(0) = \emptyset$ and $V(1) = \{a, b\}$. Here, M is a sequence model of C'_{79} but not of C_{12}.

Characterization of Pnueli and Shalev's Semantics. We now show that the step responses of a parallel configuration C, according to Pnueli and Shalev's semantics, can be characterized as particular sequence models of C, to which we refer as *response models*. The response models of C are the sequence models of C of length 1, i.e. subsets of $\Pi \setminus \{\bot\}$ that do not occur as the final world of any other sequence model of C except itself.

Definition 2. *Let $C \in \mathsf{PC}$. Then, $M = (1, V) \in SM(C)$ is a* response model *of C if $K^* = M^*$ implies $K = M$, for all $K \in SM(C)$.*

Intuitively, the validity of this characterization is founded in Pnueli and Shalev's closed-world assumption which requires a response to emerge from within the considered configuration and not by interactions with the environment.

Theorem 1. *Let $C \in \mathsf{PC}$ and $E, A \subseteq_{fin} \Pi$. Then, $C \Downarrow_E A$ iff A is a response model of $C \parallel \cdot / E$.*

Thm. 1 provides a simple model-theoretic characterization of operational step responses; e.g., configuration $\overline{a}/a$ forces Pnueli and Shalev's step-construction procedure to fail. Indeed, the only sequence model of $\overline{a}/a$ of length 1 (and using only event a) is $A = \{a\}$. But A is not a response model since it is the final world of $K = (2, V) \in SM(\overline{a}/a)$ with $V(0) =_{\mathrm{df}} \emptyset$ and $V(1) =_{\mathrm{df}} A$. Since $\overline{a}/a$ does not have any response model, it can only fail. As another example, consider $a/b \parallel b/a$ which possesses the sequence models $(2, V)$, where $V(0) =_{\mathrm{df}} \emptyset$ and $V(1) =_{\mathrm{df}} \{a, b\}$, and $(1, V')$, where $V'(0) =_{\mathrm{df}} \emptyset$. Only the latter is a response model, in accordance with causality. Thus, $(a/b \parallel b/a) \Downarrow \emptyset$ is the only response.

Full Abstraction. Sequence models also lead to a fully-abstract semantics for parallel configurations within parallel contexts.

Theorem 2. *Let $C, D \in \mathsf{PC}$. Then, $SM(C) = SM(D)$ iff $\forall R \in \mathsf{PC}\, \forall E, A \subseteq_{fin} \Pi$. $C \| R \Downarrow_E A$ iff $D \| R \Downarrow_E A$.*

Hence, sequence models contain precisely the information needed to capture all possible interactions of a parallel configuration with all potential environments.

Characterization of Sequence Models. Of course, Thm. 2 does not mean that every set of stabilization sequences can be obtained from a (parallel) configuration. In fact, in intuitionistic logic it is known that in order to specify arbitrary linear sequences, nested implications are needed [18]. Configurations, however, only use first-order implications and negations. Their sequence models may be characterized by simple lattice structures which we refer to as *behaviors*.

Definition 3. *An A-behavior $\mathcal{C}$, for $A \subseteq_{fin} \Pi$, is a pair $\langle F, I \rangle$, where $F \subseteq 2^{A \setminus \{\perp\}}$ and I is a monotonic function that maps every $B \in F$ to a set $I(B) \subseteq 2^B$ such that $B \in I(B)$ and $I(B)$ is closed under intersection, i.e., $B_1, B_2 \in I(B)$ implies $B_1 \cap B_2 \in I(B)$, for all $B \in F$. Furthermore, $\mathcal{C}$ is called* bounded, *if $A \in F$.*

It is not difficult to show that the pairs of initial and final states occurring together in the sequence models of $C \in \mathsf{PC}$ induce a behavior. More precisely, if A is the set of events mentioned in C, then the *induced* A-behavior $Beh(C)$ of C is the pair $\langle F(C), I(C) \rangle$, where

$$F(C) =_{\mathrm{df}} \{E \subseteq A \mid \exists (n, V) \in SM(C).\, V(n-1) = E\}$$
$$I(C)(B) =_{\mathrm{df}} \{E \subseteq B \mid \exists (n, V) \in SM(C).\, V(0) = E \text{ and } V(n-1) = B\}\,.$$

Note that the response models B of C are precisely those $B \in F(C)$ for which $I(C)(B) = \{B\}$. As desired, we obtain the following theorem.

Theorem 3. *$\forall C, D \in \mathsf{PC}$. $Beh(C) = Beh(D)$ iff $SM(C) = SM(D)$.*

In conjunction with Thm. 2 it is clear that equivalence in arbitrary parallel contexts can as well be decided by behaviors: $Beh(C) = Beh(D)$ iff $\forall R \in \mathsf{PC}$ $\forall E, A \subseteq_{\mathrm{fin}} \Pi$. $C \| R \Downarrow_E A$ iff $D \| R \Downarrow_E A$. In contrast to $SM(C)$, however, $Beh(C)$ provides an *irredundant* representation of parallel configurations:

Theorem 4. *$\mathcal{C}$ is a (bounded) A-behavior iff there exists a configuration $C \in \mathsf{PC}$ over events A (not using $\perp$) such that $\mathcal{C} = Beh(C)$.*

Summarizing, behaviors $Beh(C)$, where $C \in \mathsf{PC}$, yield a model representation of $SM(C)$. For each B in $F(C)$, the set $I(C)(B)$ is a $(\cap, \subseteq)$ semi-lattice with maximal element B. As a very simple example, consider $C =_{\mathrm{df}} bc/a \,\|\, ac/b \,\|\, \overline{a}/a \,\|\, \overline{b}/b \,\|\, \overline{c}/c$ over events $A = \{a, b, c\}$. Its corresponding bounded A-behavior $Beh(C)$ is depicted in Fig. 2. Since $F(C) = \{A\}$, we only have the $(\cap, \subseteq)$ semi-lattice $I(C)(A)$. Generally speaking, $SM(C)$ is the set of sequences whose world-wise intersection with A are paths in the lattice diagrams ending in maximal elements. Moreover, the maximal elements are the classic solutions of C which may become actual responses in suitable parallel contexts.

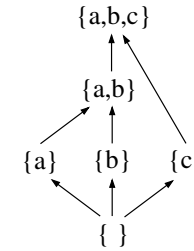


Fig. 2. Bounded $\{a, b, c\}$-behavior

4 Generalizing the Full-Abstraction Result

In this section we reduce the problem of full abstraction for arbitrary configurations in arbitrary contexts to that for parallel configurations in parallel contexts.

Reduction to Parallel Contexts. For extending the full-abstraction result to arbitrary contexts, one must address a compositionality problem for $+$ which already manifests itself in Pnueli and Shalev's semantics. Consider configurations $C =_{\mathrm{df}} \overline{a}/b$ and $D =_{\mathrm{df}} \overline{a}/b \,\|\, a/a$ which have the same responses in all parallel contexts. However, in the choice context $\Phi[x] = (\cdot/e + x) \| \cdot/a$ we have $\Phi[D] \Downarrow \{a\}$ but $\Phi[C] \not\Downarrow \{a\}$ (as $\Phi[C] \Downarrow \{a, e\}$ only). This context is able to detect that D is enabled by the environment $\cdot/a$ while C is not. Hence, one has to take into account whether there exists a transition in C that is triggered for a set A of events. To store the desired information we use the *triggering indicator* $\rho(C, A) \in \mathbb{B} =_{\mathrm{df}} \{ff, tt\}$ defined by $\rho(C, A) =_{\mathrm{df}} tt$, if $\mathsf{triggered}(C, A) \neq \emptyset$, and $\rho(C, A) =_{\mathrm{df}} ff$, otherwise.

Lemma 1. *Let $C, D \in \mathcal{C}$. Then $C \simeq D$ iff $\forall P \in \mathsf{PC}, A \subseteq_{fin} \Pi, b \in \mathbb{B}$. $(C \| P \Downarrow A$ and $\rho(C, A) = b)$ iff $(D \| P \Downarrow A$ and $\rho(D, A) = b)$.*

Thus, to ensure compositionality for arbitrary contexts we only need to record $[\![C]\!]_1^b =_{\mathrm{df}} \{\langle A, P\rangle \mid C \| P \Downarrow A,\ \rho(C, A) = b,\ P \in \mathsf{PC}\}$, for $b \in \mathbb{B}$, instead of $[\![C]\!]_0$. We may view $[\![C]\!]_1^{tt}$ as the collection of *active* and $[\![C]\!]_1^{ff}$ as the collection of *passive* responses for C in parallel contexts, according to whether a transition of C takes part in response A. By Lemma 1, $C \simeq D$ iff $[\![C]\!]_1^{tt} = [\![D]\!]_1^{tt}$ and $[\![C]\!]_1^{ff} = [\![D]\!]_1^{ff}$.

Reduction to Parallel Configurations. For eliminating the choice operator from configurations we employ a *distributivity law*. However, the naive distributivity law $C \simeq D$ for $C =_{\mathrm{df}} (t_1 + t_2) \| t_3$ and $D =_{\mathrm{df}} (t_1 \| t_3') + (t_2 \| t_3'')$, where transitions t_3' and t_3'' are identical to t_3 except for their name, does in general not hold. Consider $t_i =_{\mathrm{df}} a_i \overline{b_i}/c_i$, for $1 \leq i \leq 3$, and assume that all events are

mutually distinct. Then, in a context in which t_2 is enabled but not t_1, transition t_3 in C is forced to interact with t_2, while in D transition t_3' may run by itself in the summand $t_1 \| t_3'$. E.g., if $E = \{a_2, a_3\}$ then $D \Downarrow_E \{c_3, a_2, a_3\}$, but the only A with $c_3 \in A$ and $C \Downarrow_E A$ is $A = \{c_2, c_3, a_2, a_3\}$.

The naive distributivity law can be patched by adding configurations $D_1(t_3)$ and $D_2(t_3)$ such that $C \simeq t_1 \| D_1(t_3) + t_2 \| D_2(t_3)$. Here, $D_i(t_3)$ must weaken t_3 such that it disables t_3, whenever t_i is not enabled but t_{3-i} is. A simple way to achieve this is to define $D_1(t_3) =_{\mathrm{df}} D_1 \| t_3'$ and $D_2(t_3) =_{\mathrm{df}} D_2 \| t_3''$, where $D_i =_{\mathrm{df}} \overline{a}_i a_{3-i} \overline{b}_{3-i} / \bot \parallel b_i a_{3-i} \overline{b}_{3-i} / \bot$, for $i \in \{1, 2\}$. As desired, the "watchdog" configuration D_i satisfies for all parallel contexts P: $D_i \| P \Downarrow A$ iff (i) $P \Downarrow A$ and (ii) A triggers t_i or does not trigger t_{3-i}. It should be clear how this can be generalized, i.e., how one constructs for any $C, D \in \mathsf{C}$ a configuration $\mathsf{watch}(C, D)$ such that $P \| \mathsf{watch}(C, D) \Downarrow A$ iff (i) $P \Downarrow A$ and (ii) $\mathsf{triggered}(C, A) \neq \emptyset$ or $\mathsf{triggered}(D, A) = \emptyset$.

Lemma 2. *Let $C_1, C_2, D \in \mathsf{C}$. Then, $(C_1 + C_2) \| D \simeq (\mathsf{watch}(C_1, C_2) \| C_1 \| D) + (\mathsf{watch}(C_2, C_1) \| C_2 \| D)$.*

The fact that we have available an explicit failure event $\bot$ makes this distributivity law particularly simple. The use of $\bot$, however, is inessential as it can be eliminated [12]. Now, by repeatedly applying distributivity we may push occurrences of operator $+$ to the outside of configurations.

Lemma 3. *Let $C \in \mathsf{C}$. Then, there exists a finite index set $\mathsf{ind}(C)$ and $C_i \in \mathsf{PC}$, for $i \in \mathsf{ind}(C)$, such that $C \simeq \sum_{i \in \mathsf{ind}(C)} C_i$.*

Hence, $[\![C]\!]_1 = [\![\sum_{i \in \mathsf{ind}(C)} C_i]\!]_1$. Moreover, since an active response of a sum must be an active response of *one* of its summands and since a passive response of a sum always is a passive response of *all* of its summands, $[\![\sum_{i \in \mathsf{ind}(C)} C_i]\!]_1^{tt} = \bigcup_{i \in \mathsf{ind}(C)} [\![C_i]\!]_1^{tt}$ and $[\![\sum_{i \in \mathsf{ind}(C)} C_i]\!]_1^{ff} = \bigcap_{i \in \mathsf{ind}(C)} [\![C_i]\!]_1^{ff}$ hold. Thus, we obtain:

Lemma 4. *Let $C, D \in \mathsf{C}$. Then, $C \simeq D$ iff $\bigcup_{i \in \mathsf{ind}(C)} [\![C_i]\!]_1^{tt} = \bigcup_{j \in \mathsf{ind}(D)} [\![D_j]\!]_1^{tt}$ and $\bigcap_{i \in \mathsf{ind}(C)} [\![C_i]\!]_1^{ff} = \bigcap_{j \in \mathsf{ind}(D)} [\![D_j]\!]_1^{ff}$.*

Full-abstraction Result. Now, we are able to use our analysis of Sec. 3 to phrase Lemma 4 in terms of behaviors. All we need to do is to replace the parallel configuration $P \in \mathsf{PC}$ in every pair $\langle A, P \rangle \in [\![C_i]\!]_1$, for $i \in \mathsf{ind}(C)$, by its behavior $Beh(P)$. It turns out that the pairs obtained in this way can be uniquely determined from the behavior $Beh(C_i)$ of C_i, for any $i \in \mathsf{ind}(C)$.

Definition 4. *Let $A \subseteq_{fin} \Pi$. An A-behavior $\langle F, I \rangle$ is called an A-context for $C \in \mathsf{PC}$ if (i) $F = \{A\}$, (ii) $A \in F(C)$, and (iii) $I(A) \cap I(C)(A) = \{A\}$.*

Note that A-contexts for C are bounded behaviors, i.e., they can be represented without $\bot$. An A-context $\mathcal{P}$ of C represents a set of sequences that all end in the final world A, in which also some sequence model of C must end and which only have world A in common with the sequence models of C ending in A. These properties imply $C \| P \Downarrow A$, for every P with $Beh(P) = \mathcal{P}$. Hence, A-contexts $\mathcal{P}$ are "relativized complements" of C wrt. the final response A.

Consider again example C from above, whose sequence models $SM(C)$ are described by the behavior of Fig. 2. To get the A-contexts of C, where $A = \{a, b, c\}$, we must take the "complement" of $I(C)(A)$, i.e., all $B \subset A$ that are missing in the lattice of Fig. 2. As shown in Fig. 3, C has two A-contexts $\mathcal{P}_1$ and $\mathcal{P}_2$ covering this complement; configurations that denote them are $P_1 =_{\mathrm{df}} \cdot/ac \,\|\, \overline{b}/b$ and $P_2 =_{\mathrm{df}} \cdot/bc \,\|\, \overline{a}/a$, respectively. These provide complete information since every A-context must be contained in $\mathcal{P}_1$ or $\mathcal{P}_2$. For all $C \in \mathsf{PC}$ and $b \in \mathbb{B}$ we are finally led to define $[\![C]\!]_2^b =_{\mathrm{df}} \{\langle A, \mathcal{P}\rangle \mid A \subseteq_{\mathrm{fin}} \Pi,\ \rho(C, A) = b,\ \mathcal{P}$ is A-context of $C\}$ and obtain as a corollary to Lemma 4 and Thm. 2:

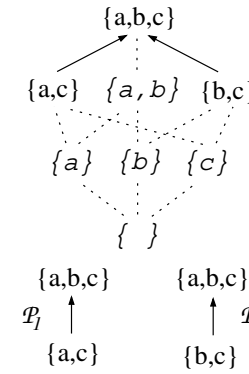


Fig. 3. $\{a, b, c\}$-contexts

Theorem 5. *Let $C, D \in \mathsf{C}$. Then, $C \simeq D$ iff $\bigcup_{i \in \mathit{ind}(C)} [\![C_i]\!]_2^{tt} = \bigcup_{j \in \mathit{ind}(D)} [\![D_j]\!]_2^{tt}$ and $\bigcap_{i \in \mathit{ind}(C)} [\![C_i]\!]_2^{ff} = \bigcap_{j \in \mathit{ind}(D)} [\![D_j]\!]_2^{ff}$.*

With Thm. 5 we have finally achieved our goal, as $[\![C]\!]_2$ is satisfactorily semantical and finite. In combination with Lemma 2 it directly lends itself to be applied for a model-based implementation of Pnueli and Shalev's semantics, which does not require backtracking for handling failure. Finally, it should be stressed that the above theorem also holds if we restrict ourselves to "+"-configurations of the form $C + t$, as in Statecharts, instead of permitting configurations $C + D$, for arbitrary $C, D \in \mathsf{C}$ (cf. Sec. 2). We now return to the example of Figs. 2 and 3. Let id_B be the behavior $\langle \{B\}, B \mapsto \{B\}\rangle$, for $B \subseteq \Pi$. We have $[\![C]\!]_2^{tt} = \{\langle \{a, b, c\}, \mathcal{P}_i\rangle \mid i = 1, 2\}$ and $[\![C]\!]_2^{ff} = \emptyset$. The same semantics can be generated as $D_1 + D_2$, where $D_1 = bc/a \,\|\, \overline{b}/b \,\|\, \overline{a}/a$ and $D_2 = ac/b \,\|\, \overline{b}/b \,\|\, \overline{c}/c$, since $[\![D_i]\!]_2^{tt} = \{\langle \{a, b, c\}, \mathcal{P}_i\rangle\}$, $[\![D_1]\!]_2^{ff} = \{\langle \{a, b\}, \mathrm{id}_{\{a,b\}}\rangle\}$, $[\![D_2]\!]_2^{ff} = \{\langle \{b, c\}, \mathrm{id}_{\{b,c\}}\rangle\}$. Hence, $[\![D_1]\!]_2^{tt} \cup [\![D_2]\!]_2^{tt} = [\![C]\!]_2^{tt}$ and $[\![D_1]\!]_2^{ff} \cap [\![D_2]\!]_2^{ff} = \emptyset = [\![C]\!]_2^{ff}$. By Thm. 5, $C \simeq D_1 + D_2$. A similar reasoning reveals $C_{79} \simeq C'_{79}$ (cf. Sec. 3).

5 Discussion and Related Work

Our investigation focused on Pnueli and Shalev's presentation of Statecharts and its macro-step semantics. The elegance of their operational semantics manifests itself in the existence of an equivalent *declarative fixed point semantics* [16]. However, as illustrated in [16], this equivalence is violated when allowing disjunctions in transition triggers. For example, the configurations $(\overline{a} \vee b)/a$ and $\overline{a}/a \,\|\, b/a$ do not have the same response behavior. This subtlety can now be explained in our framework. In Pnueli and Shalev's setting, $\overline{a} \vee b$ is classically interpreted as "throughout a macro step, not a or b." In contrast, this paper reads the configuration as "throughout a macro step not a or throughout b."

Our framework can also be employed for analyzing various other variants of Statecharts semantics, such as the one of Maggiolo-Schettini et al. [14] which in

turn is inspired by the process-algebraic semantics presented in [17]. In [14] the step-construction procedure cannot fail since a transition is only considered to be enabled, if it is enabled in the sense of Pnueli and Shalev *and* if it does not produce any event that violates global consistency. As an example, consider the configuration $C =_{\mathrm{df}} t_1 \| t_2$, where $t_1 =_{\mathrm{df}} a/b$ and $t_2 =_{\mathrm{df}} \bar{b}/a$. According to [14], when C is evaluated for the empty environment, response $\{a\}$ is obtained; in Pnueli and Shalev's semantics, however, the step construction fails. The difference can be explained in terms of stabilization sequences. While Pnueli and Shalev take t_1 to stand for the specification $a \supset b$ and t_2 for $\neg b \supset a$, Maggiolo-Schettini et al. apply the interpretation $a \supset (b \vee \neg b)$ for t_1 and $\neg b \supset (a \vee \neg a)$ for t_2. Indeed, as one verifies, $\{a\}$ then is a response model of $t_1 \| t_2$. Note again that $a \vee \neg a$ is different from *true* in intuitionistic logic. Generalizing this example, the transition semantics of [14] can be captured in terms of response models by reading a transition E/A as formula $E \supset (A \vee \neg A)$, if our setting would be extended to allowing disjunctions as part of actions.

Our intuitionistic approach is also related to recent work in *synchronous languages*, especially for Berry's *Esterel* [3]. In Esterel, causality was traditionally treated separately from compositionality and synchrony as part of type-checking specifications. If the (conservative) type checker found causality to be violated, it rejected the specification under consideration. Otherwise, the specification's semantics could be determined in a very simple fashion; one may — in contrast to Statecharts semantics — abstract from the construction details of macro steps while preserving compositionality, as shown by Broy in [4]. Version 5 of Esterel [2] replaced the treatment of causality by defining a semantics via a particular Boolean logic that is *constructive*, as is intuitionistic logic.

Denotational semantics and full abstraction were also studied by Huizing et al. [10,11] for an early and later-on rejected Statecharts semantics [9]. That semantics does not consider global consistency, which makes their result largely incomparable to ours. Finally, it should be mentioned that the lack of compositionality of Statecharts semantics inspired the development of new languages, such as Alur et al.'s *communicating hierarchical state machines* [1].

6 Conclusions

To the best of our knowledge, this is the first paper to present a fully-abstract Statecharts semantics for Pnueli and Shalev's original macro-step semantics [16]. The latter semantics is non-compositional as it employs classic logic for interpreting macro steps. In contrast, our semantics borrows ideas from intuitionistic logic. It encodes macro steps via stabilization sequences which we characterized using semi-lattice structures, called behaviors. Behaviors capture the interactions between Statecharts and their environments and consistently combine the notions of causality, global consistency, and synchrony. Moreover, our approach suggests a model-based implementation of Pnueli and Shalev's semantics, thereby eliminating the need to implement failure via backtracking.

Acknowledgments. This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, Virginia 23681-2199, USA. We would also like to thank the members of the Sheffield Verification and Testing Group and our anonymous referees for their valuable comments and suggestions.

References

1. R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. In *ICALP '99*, volume 1644 of *LNCS*, pages 169–178, 1999.
2. G. Berry. The constructive semantics of pure Esterel, 1999. Draft Version 3. Available at http://www-sop.inria.fr/meije/Personnel/Gerard.Berry.html.
3. G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *SCP*, 19(2):87–152, 1992.
4. M. Broy. Abstract semantics of synchronous languages: The example Esterel. Technical Report TUM-I9706, Munich Univ. of Technology, 1997.
5. W. Damm, B. Josko, H. Hungar, and A. Pnueli. A compositional real-time semantics of STATEMATE designs. In *Compositionality: The Significant Difference*, volume 1536 of *LNCS*, pages 186–238, 1997.
6. D. Harel. Statecharts: A visual formalism for complex systems. *SCP*, 8:231–274, 1987.
7. D. Harel and E. Gery. Executable object modeling with Statecharts. *IEEE Computer*, pages 31–42, July 1997.
8. D. Harel and A. Naamad. The STATEMATE semantics of Statecharts. *ACM Trans. on Software Engineering*, 5(4):293–333, 1996.
9. D. Harel, A. Pnueli, J. Pruzan-Schmidt, and R. Sherman. On the formal semantics of Statecharts. In *LICS '87*, pages 54–64. IEEE Computer Society Press, 1987.
10. C. Huizing. *Semantics of Reactive Systems: Comparison and Full Abstraction*. PhD thesis, Eindhoven Univ. of Technology, 1991.
11. C. Huizing, R. Gerth, and W.-P. de Roever. Modeling Statecharts behavior in a fully abstract way. In *CAAP '88*, volume 299 of *LNCS*, pages 271–294, 1988.
12. G. Lüttgen and M. Mendler. What is in a step: A fully-abstract semantics for Statecharts macro steps via intuitionistic Kripke models. Technical Report CS-00-04, Univ. of Sheffield, 2000.
13. G. Lüttgen, M. von der Beeck, and R. Cleaveland. Statecharts via process algebra. In *CONCUR '99*, volume 1664 of *LNCS*, pages 399–414, 1999.
14. A. Maggiolo-Schettini, A. Peron, and S. Tini. Equivalences of Statecharts. In *CONCUR '96*, volume 1119 of *LNCS*, pages 687–702, 1996.
15. F. Maraninchi. Operational and compositional semantics of synchronous automaton compositions. In *CONCUR '92*, volume 630 of *LNCS*, pages 550–564, 1992.
16. A. Pnueli and M. Shalev. What is in a step: On the semantics of Statecharts. In *TACS '91*, volume 526 of *LNCS*, pages 244–264, 1991.
17. A.C. Uselton and S.A. Smolka. A compositional semantics for Statecharts using labeled transition systems. In *CONCUR '94*, volume 836 of *LNCS*, pages 2–17, 1994.
18. D. van Dalen. Intuitionistic logic. In *Handbook of Philosophical Logic*, volume III, chapter 4, pages 225–339. Reidel, 1986.

Algebraic Models for Contextual Nets*

Roberto Bruni[1] and Vladimiro Sassone[2]

[1] Dipartimento di Informatica, Università di Pisa, Italia.
[2] Dipartimento di Matematica e Informatica, Università di Catania, Italia.
bruni@di.unipi.it, vs@dmi.unict.it

Abstract. We extend the algebraic approach of Meseguer and Montanari from ordinary place/transition Petri nets to *contextual* nets, covering both the *collective* and the *individual* token philosophy *uniformly* along the two interpretations of net behaviors.

Introduction

Among the models for concurrency, *place/transition Petri nets* (PT nets), introduced by Petri in [14] (see also [15]), are one of the most largely diffused, with many interdisciplinary applications. The reasons of the success of the net model probably reside in the simple formal description and natural characterization of *concurrent* and *distributed systems*: the state of a system consists of a (multi)set of distributed resources, its actions consume some of the resources available and release fresh resources, thus affecting only local subsystems. In particular, a computation can be described as a partial order of events such that two events in the same computation are either causally dependent – when one could not have been executed without a resource provided by the other – or concurrent – when they could have happened in any order, because they affect independent subsystems.

Several extensions of the basic net paradigm have been considered in the literature that either increase the expressive power or give a better representation of existing phenomena. This paper focuses on *contextual nets*, also known as nets with *read arcs*, or *condition arcs*, or *test arcs* [4,13,8,21]. The underlying idea is that of *reading* resources *without consuming* them, thus providing a way of modeling multiple concurrent accesses to the same resource. With ordinary PT nets such readings must be rendered as *self-loops*, and this imposes an unfortunate sequentialization of concurrent readings. On the contrary, with contextual nets, besides pre and post-sets transitions also have '*contexts*', that is resources that are necessary for the enabling, but *not* affected by the firing. Contextual nets have found applications e.g., to transaction serializability in databases [16], concurrent constraint programming [12], and asynchronous systems [20].

The extensive use of PT nets has given rise to different schools of thought concerning their semantic interpretation. In particular, the main distinction is drawn between *collective* and *individual token philosophies* (see e.g. [19]).

* Research supported by CNR Integrated Project *Metodi per la Verifica di Sistemi Eterogenei*; by Esprit Working Groups *CONFER2* and *COORDINA*; and by MURST project *TOSCA*.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 175–186, 2000.
© Springer-Verlag Berlin Heidelberg 2000

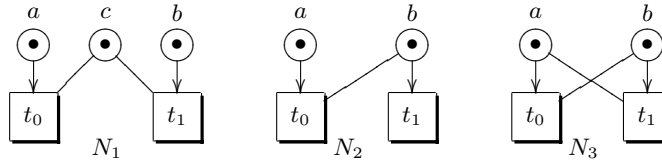


Fig. 1.

According to the collective token philosophy (*CTph*), one should not distinguish among different *tokens* in the same place (i.e., among instances of the same resource), because all such tokens are operationally equivalent. This view disregards that tokens may have different origins and histories and may, therefore, carry different *causality* information. Selecting one instance rather than another, can make the difference from being causally dependent or not on some previous event. And this may well be a piece of information one does not want to discard, which is the point of the individual token philosophy (*ITph*). Of course, causal dependencies may influence the degree of concurrency in computations, and therefore *CTph* and *ITph* lead to different concurrent semantics.

Independently of *CTph* and *ITph*, for contextual nets several different approaches have been proposed that differ for the way in which contexts are read. For example, let us consider the nets N_1, N_2 and N_3 in Figure 1 taken from [21]. (As usual, places are represented by circles, tokens by black bullets, transitions by boxes, pre- and post-sets by directed weighted arcs, and contexts by undirected weighted arcs, with unary weights always omitted.) According to [13], the transitions t_0 and t_1 can fire concurrently in N_1, but neither in N_2 nor in N_3, since the basic assumption is that a token cannot be read and consumed in the same step. In [8], instead, the concurrent step is allowed for all three nets, the basic assumption being that t_0 and t_1 can both start together and read the context tokens, without needing them while the actions take place. Besides its possible merits, we find this interpretation not fully convincing as, for instance, in N_3 we would end up in a state that cannot be reached by any firing sequence. The basic assumption of [21] that firings have duration leads to consider ST-traces, where explicit *transition-starts* and *transition-ends* events are fired. Hence N_2 can start t_0 and then t_1 before t_0 completes, allowing the concurrent step $\{t_0, t_1\}$. On the contrary, in N_3 if either t_0 or t_1 starts, then the context for the other transition is consumed and the concurrent step is forbidden. In this paper, we follow the interpretation of [13] that fits better our understanding of contexts.

Collecting Tokens. The seminal paper [10] proposed an algebraic approach to the analysis of net behaviors relying on the basic observation that the monoidal structure of PT net states (i.e., the *markings*) can be lifted to the level of computations so to obtain an algebraic initial model for concurrent net behaviors according to the *CTph*.

The algebraic net theory developed under the *CTph* is well consolidated, and the relationships between its *computational*, *algebraic* and *logical* interpretations are by now very clear [3]. Starting with the classical 'token-game' semantics, many behavioral models for Petri nets have been proposed that follow the *CTph*. In particular, the *commutative processes* of Best and Devillers [2] reconcile the 'diamond' equivalence on firing and step sequences, and express very nicely the concurrency of the model. They also admit an exact algebraic representation by means of the universal construction $\mathcal{T}(_)$ that yields *strictly symmetric strict monoidal categories* from the category of PT nets. More precisely, given a PT net N, the objects of $\mathcal{T}(N)$ are the elements of the free commutative monoid over the set of places, its arrows correspond to the commutative processes of N [10,5].

Surprisingly, the *CTph* semantics for contextual nets have received poor attention in the literature. Whether because the problem has been underestimated, or simply because the *ITph* is more fascinating, we cannot tell. In any case, we think that it is useful to remove this discrepancy with the semantics of ordinary PT nets. Moreover, although one can easily extend the diamond equivalence to firing sequences on contextual nets, the formalization of a good *algebraic* model is not at all straightforward. Inspired by a suggestion made by Meseguer in [9], we give here a fully satisfactory treatment of this issue. The idea is to consider monoidal categories with a commutative tensor product taken – differently from the case of PT nets – over a *non-free* monoid of places. In particular, we regard each token a as an *atom* that can emit several 'negative' *particles* a^-, while keeping track of the number of electrons around, i.e., as in [9], we assume that for all $k \in \mathbb{N}$, $a = a^k \otimes k \cdot a^-$, with a^k a shorthand for $a^{+\cdots+}$ ($+$ applied k times).

Replacing context arcs on a with self-loop arcs on a^-, we are able to give an axiomatic construction of a monoidal category whose arrows between standard markings (i.e., containing no negative particles) are (isomorphic to) the concurrent computations of the net according to the *CTph*. A key ingredient for this result to hold is the so-called *maximum sharing hypothesis*, an axiom expressing that concurrent readings can *always* be seen as sharing the same token, a fundamental idea in *CTph*.

Observing Causal Dependencies. Building on the notion of *process* introduced by Goltz and Reisig in [7], several authors have shown that the semantics of nets in the *ITph* can still be understood in terms of *symmetric monoidal categories*. In particular, a simple variation of Goltz-Reisig processes called *concatenable processes* is introduced in [5] (see also [17]), which admits sequential composition and yields a symmetric monoidal category $\mathcal{P}(N)$ for each net N. Also several unfolding semantics (see e.g. [22,11]) have been proposed that give a denotational interpretation of the interplay between concurrency, causality and nondeterminism.

For contextual nets both the process and the unfolding approaches have been studied [13,1], giving a satisfactory understanding of the computational model via the introduction of *asymmetric event structures*. The algebraic approach, however, has been pursued only in a recent paper by Gadducci and Montanari [6] using *match-share* categories. There, the basic idea is that, together

with symmetries, two additional auxiliary constructors must be present: one for *duplicating* tokens and one for *matching* them. Read arcs can then be replaced by self-loops, and reading without consuming modeled by duplicating the context, firing the transition concurrently with an idle copy of the context, and then matching the idle copy with the corresponding produced tokens. Multiple concurrent access is achieved by producing via duplication – and then absorbing via matching – enough copies of the context. In [6] a suitable axiomatization of duplicators and matchers is introduced and proved to represent faithfully the basic fact about concurrent access: steps sharing the same context, but otherwise disjointly enabled, can execute concurrently or in any interleaved order with *no* noticeable difference. The main problem of this approach is that the initial model contains *too many* arrows and, therefore, in order to obtain a bijection with contextual processes one has to carve a suitable subcategory. Although the arrows of this subcategory can be characterized by inspecting their structure, the lack of a global correspondence somehow weakens the framework.

We aim at improving the approach of [6], by noticing that unwanted arrows are due to *redundant* information in the model. In fact, once a context token is read by a transition we know the 'real' token it is connected to: the one duplication was applied to. Hence, the match operation, needed for expressing concurrent readings, does not add any further information and may introduce inconsistent behaviors. For example, given two tokens in the place a, one can first duplicate both and then match each copy of the first token with a copy of the second token: The resulting arrow is meaningless from the computational viewpoint. We overcome this problem by extending to the ITph the approach proposed in the first part of the paper for the CTph. In particular, besides a^+ and a^- we introduce the term a_- for each place a, with $a = a^k \otimes k \cdot a_-$.

Each context arc from a to t is then replaced by putting a^- in the source of t and a_- in the target of t. This is necessary to avoid that contexts released by a transition be consumed by another transition, and represents, in the ITph, a sort of dual to the maximum sharing hypothesis. Then, we introduce symmetries on markings, but *regulate* their use on the a^+, a^- and a_- as to *forbid* the swapping of a a^+ and an adjacent a^- or a_-. This is actually the key of our proposal, as it prevents that electrons may migrate from atom to atom, which is essentially what happens in [6]. We impose this restriction by omitting the corresponding symmetries. Putting such arrows back in the model would in fact result in a redundant framework perfectly analogous to the one of match-share categories. Our main result is that, again, the arrows between standard markings are in bijection with a slight refinement of contextual processes, called *strongly concatenable*.

Structure of the Paper. In Section 1 we recall some basics about contextual nets and the algebraic semantics of PT nets. In Sections 2 and 3 we define algebraic semantics for contextual nets under both the CTph and the ITph, providing original characterization results for commutative and strongly concatenable contextual processes. We remark that in the absence of read arcs, our semantics coincide with the classical ones.

1 Preliminaries

1.1 Contextual Nets

Contextual nets were introduced for extending PT nets with the 'read without consume' operation [4,13,8,21]. The states of contextual nets are called *markings* and represent distributions of resources (*tokens*) in typed repositories (*places*). Given the set of places S, markings can be seen as multisets $u\colon S \to \mathbb{N}$, where $u(a)$ denotes the number of tokens that place a carries in u. The set of finite multiset on S if a free commutative monoid on S. We denote it by $S^{\oplus}$, and indicate multiset inclusion, difference and union by $\subseteq$, $\oplus$ and $\ominus$, respectively. For k a natural number and u a multiset, $k \cdot u$ is the multiset such that $(k \cdot u)(a) = k \cdot u(a)$ for all a. We denote by $\lfloor u \rfloor$ the underlying set of u, that can be seen as the multiset such that $\lfloor u \rfloor(a) = 1$ if $u(a) > 0$ and $\lfloor u \rfloor(a) = 0$ otherwise.

Definition 1. *A* contextual net N *is a tuple* $(S, T, \partial_0, \partial_1, \varsigma)$*, where* S *is the set of places,* T *is the set of transitions,* $\partial_0, \partial_1\colon T \to S^{\oplus}$ *are the pre and post-set functions, and* $\varsigma\colon T \to S^{\oplus}$ *is the context function.*

Informally, $\partial_0(t) \oplus \varsigma(t)$ is the minimum amount of resources that t requires to be enabled. Of these resources, those in $\partial_0(t)$ are retrieved and consumed, while those in $\varsigma(t)$ are just read and left on their repositories. When t has accomplished its task, it returns $\partial_1(t)$ fresh tokens and releases the context. Only at this point other transitions will be able to consume the tokens in $\varsigma(t)$, whereas they can use the same context concurrently with t. Besides the usual assumption that $\varsigma(t)$ and $\partial_0(t) \oplus \partial_1(t)$ are disjoint for each transition t, we assume that $\varsigma(t)$ is a *set*.

Definition 2. *Let* u *and* v *be markings, and* X *a finite multiset of transitions of a contextual net* $N = (S, T, \partial_0, \partial_1, \varsigma)$*. We say that* u *evolves to* v *under the* step X*, in symbols* $u\ [X\rangle\ v$*, if the transitions in* X *are concurrently enabled at* u*, i.e.,* $\lfloor \bigoplus_{t \in T} \varsigma(t) \rfloor \oplus \bigoplus_{t \in T} X(t) \cdot \partial_0(t) \subseteq u$*, and*

$$v = u \ominus \left(\bigoplus_{t \in T} X(t) \cdot \partial_0(t) \right) \oplus \bigoplus_{t \in T} X(t) \cdot \partial_1(t).$$

A step sequence *from* u_0 *to* u_n *is a sequence* $u_0\ [X_1\rangle\ u_1 \ldots u_{n-1}\ [X_n\rangle\ u_n$.

Thus the execution of the step X requires that the marking u contains at least *all* the tokens in the preconditions of transitions in X plus at least *one* token for each place that is used as context by some transition in X. This matches the intuition that a token can be used as context by many transitions at the same time. From the concurrent point of view, the fact that transitions in X are executed in a step means that they can be equivalently executed in any order. Thus, likewise ordinary PT nets, step sequences for contextual nets can be considered up to the equivalence induced by the *diamond transformation* relation $_ \diamond _$ defined by $u\ [X \oplus Y\rangle\ v \diamond u\ [X\rangle\ u_1\ [Y\rangle\ v$ for any step $u\ [X \oplus Y\rangle\ v$ (and suitable u_1). The diamond equivalence is the reflexive, symmetric, transitive and sequences concatenation closure of the relation $_ \diamond _$.

Definition 3. *Given a contextual net N, the strictly symmetric strict monoidal category of* contextual commutative processes $\mathcal{CCP}(N)$ *has the markings of N as objects and its step sequences, taken modulo the diamond equivalence, as arrows.*

In the *ITph* computations are commonly described in terms of structures representing the causal relationships between event occurrences. In the case of nets, this is fruitfully formalized through the following notion of process. We remark that these notions are conservative extension of the corresponding notions for ordinary PT nets, to which they reduce in the absence of read arcs.

Definition 4. *A* contextual process net *is a finite, acyclic (w.r.t. the preorder in which t precedes t' if either $\partial_1(t) \cap (\partial_0(t') \cup \varsigma(t')) \neq \varnothing$ or $\varsigma(t) \cap \partial_0(t') \neq \varnothing$) contextual net Θ such that (1) for all $t \in T_\Theta$, $\partial_0(t)$ and $\partial_1(t)$ are sets (as opposed to multisets), and (2) for all pairs $t_0 \neq t_1 \in T_\Theta$, $\partial_i(t_0) \cap \partial_i(t_1) = \varnothing$, for $i = 0, 1$.*

Definition 5. *A* contextual process *π of a contextual net N consists of a contextual process net Θ together with a pair of functions $\langle \pi_T, \pi_S \rangle$, where $\pi_T \colon T_\Theta \to T_N$ and $\pi_S \colon S_\Theta \to S_N$, that respect source, target and context, i.e., such that $\partial_{N\,i} \circ \pi_T = \pi_S \circ \partial_{\Theta\,i}$, for $i = 0, 1$, and $\varsigma_N \circ \pi_T = \pi_S \circ \varsigma_\Theta$. Contextual processes are considered up to isomorphisms.*

1.2 Petri Nets Are Monoids

The paper [10] exploited the monoidal structure of markings to provide an algebraic characterization of the concurrent computations of nets. The basic idea was to lift the structure of states to the level of transitions, providing an algebraic representation of concurrent firing. In turn, these 'algebraic' steps can be sequentially concatenated in order to express more complex computations. Since sequential composition endows computations with a categorical structure – markings are objects, computations are arrows, and idle tokens are identities – the parallel composition yields a tensor product. The interplay of parallel and sequential composition regulated by functoriality of tensor products models a basic fact about concurrency, namely that concurrent transitions can occur in any order. Under the *CTph* the tensor product can simply be commutative. Then, each PT net N freely generates a strictly symmetric strict monoidal category $\mathcal{T}(N)$ whose arrows are in bijection with the *commutative processes* of N [2].

Under the *ITph* the situation is more complex. In order to be able to model causal dependencies one cannot consider multisets of transitions. The proposal of Degano, Meseguer and Montanari was to introduce a non commutative tensor product – while keeping markings as objects – together with suitable arrows for exchanging the order in which transitions fetch and produce tokens [5]. Such arrows are called symmetries, and are formalized categorically as the components of a natural isomorphism. This approach leads to the construction of a (non strictly) symmetric strict monoidal category $\mathcal{P}(N)$ for each net N, whose arrows define the *concatenable processes* of N. A more concrete construction, $\mathcal{Q}(N)$, was introduced in [18] in order to remove some deficiencies of the previous approach. The main feature of $\mathcal{Q}(N)$, which captures the so-called *strongly concatenable processes*, is that its objects are strings rather than multisets of tokens.

(1)	$(u)^{+} \oplus (u)^{-} = u$	(5)	$((u)^{-})^{-} = \varnothing$
(2)	$((u)^{-})^{+} = (u)^{-}$	(6)	$((u)^{+})^{-} = (u)^{-}$
(3)	$(u \oplus v)^{+} = (u)^{+} \oplus (v)^{+}$	(7)	$(u \oplus v)^{-} = (u)^{-} \oplus (v)^{-}$
(4)	$(\varnothing)^{+} = \varnothing$	(8)	$(\varnothing)^{-} = \varnothing$

Fig. 2.

2 Collective Contexts

As explained in the Introduction, we build the algebraic theory over a non-free monoid of places. In particular, apart from the commutative monoidal operation $_ \oplus _$ with unit $\varnothing$, we consider other two operations $(_)^{+}$ and $(_)^{-}$ that are axiomatized as in Figure 2, where we omit the usual associativity, commutativity and unit axioms for $_ \oplus _$. These mean precisely that $(_)^{+}$ and $(_)^{-}$ are monoid homomorphisms such that $(_)^{+} \oplus (_)^{-} = id$, $(_)^{+} \circ (_)^{-} = (_)^{-}$, and $(_)^{-} \circ (_)^{-} = \varnothing$. Observe that (6) actually follows from (1), (7) and (5).

By these laws we can always eliminate consecutive applications of $(_)^{+}$ and $(_)^{-}$, except for sequences of $(_)^{+}$. We shall write u^k as a shorthand for $(_)^{+}$ applied k times to u and omit the parentheses. We assume $u^0 = u$, but we remark that in general $u^{+} = u^1 \neq u$. We call *molecules* the elements of this algebra. Given a set S, we let $\mu(S)$ denote the set of molecules generated by S.

Lemma 1. *For any natural number k and molecule u we have $(u^k)^{-} = u^{-}$.*

Proof. By induction, applying law (6).

Proposition 1. *For any natural k, and molecule u, we have $u^k = u^{k+1} \oplus u^{-}$*

Proof. By law (1), we have $u^k = (u^k)^{+} \oplus (u^k)^{-}$, and $(u^k)^{-} = u^{-}$ by Lemma 1.

Corollary 1. *For any natural k and molecule u, we have $u = u^k \oplus k \cdot u^{-}$.*

Of course we are interested in molecules centered on the places, these can be of two forms, either a^k or a^{-}. From the computational point of view, the a^{-} are the basic contexts, which carry very little information, since the nucleus a^k can produce as many of them as needed. To understand this point, one can think of the tokens as *ticket rolls* with unbounded number of tickets available. Readers just take a ticket and return it after the use for recycle, whereas consumers must retrieve the entire roll.

Definition 6. *Given a contextual net $N = (S, T, \partial_0, \partial_1, \varsigma)$, we define the category $\mathcal{M}(N)$ as the category with objects the molecules on S and with arrows generated from the rules in Figure 3, modulo the axioms of strictly symmetric strict monoidal categories in Figure 4.*

We can now characterize contextual commutative processes algebraically.

$$\frac{u \in \mu(S)}{id_u\colon u \to u} \qquad \frac{t \in T,\ \partial_0(t) = u,\ \partial_1(t) = v,\ \varsigma(t) = w}{t\colon u \oplus w^- \to v \oplus w^-}$$

$$\frac{\alpha\colon u \to v,\ \beta\colon v \to w}{\alpha;\beta\colon u \to w} \qquad \frac{\alpha\colon u \to v,\ \beta\colon w \to z}{\alpha \oplus \beta\colon u \oplus w \to v \oplus z}$$

Fig. 3.

$$\begin{array}{ccc}
\alpha;(\beta;\gamma)=(\alpha;\beta);\gamma & \alpha;id_v=id_u;\alpha=\alpha & \\
\alpha\oplus(\beta\oplus\gamma)=(\alpha\oplus\beta)\oplus\gamma & \alpha\oplus\beta=\beta\oplus\alpha & \alpha\oplus id_\varnothing=\alpha \\
(\alpha;\beta)\oplus(\gamma;\delta)=(\alpha\oplus\gamma);(\beta\oplus\delta) & id_{u\oplus v}=id_u\oplus id_v &
\end{array}$$

Fig. 4.

Theorem 1. *The category $\mathcal{CCP}(N)$ is isomorphic (via a monoidal functor) to the full subcategory of $\mathcal{M}(N)$ whose objects are $S_N^{\oplus}$.*

A very important property needed in the proof is what we call the *maximum sharing hypothesis*, that can be expressed as below. This result contains the core of the *CTph*, since it shows that whenever two or more tokens in the same place a are used as contexts, we can always find an equivalent computation where only one token is used (twice or more) as context.

Proposition 2. *For any molecule u and natural numbers k and n, we have $u^n \oplus u^k = u^{n+k} \oplus u$.*

Proof. By Corollary 1, we have $u^{n+k} \oplus u = u^{n+k} \oplus u^k \oplus k \cdot u^-$. By commutativity (and associativity) of $_ \oplus _$ we get $u^{n+k} \oplus u = u^{n+k} \oplus k \cdot u^- \oplus u^k$. By applying Proposition 1 k times we have the result.

For instance, let us consider the net N in Figure 5. In $\mathcal{M}(N)$ we have three basic arrows $t_0\colon a \oplus c^- \to c^-$, $t_1\colon b \oplus c^- \to c^-$ and $t_2\colon c \to \varnothing$, but neither t_0, nor t_1 can represent a commutative contextual process, since their sources and targets are not elements of $S^{\oplus}$. To remedy this, we must put t_0 and t_1 in an environment where the c^- become instances of a 'complete' token, as $id_{c^+} \oplus t_0\colon a \oplus c \to c$ and $id_{c^+} \oplus t_1$. The concurrent execution of t_0 and t_1 with shared context is instead written as $id_{c^2} \oplus t_0 \oplus t_1$. By the functoriality of $_ \oplus _$, we have that $id_{c^2} \oplus t_0 \oplus t_1 = (id_{c^+} \oplus t_0 \oplus id_b);(id_{c^+} \oplus t_1) = (id_{c^+} \oplus t_1 \oplus id_a);(id_{c^+} \oplus t_0)$, (recall that $id_{c^2} \oplus id_{c^-} = id_{c^+}$), i.e., t_0 and t_1 can execute in any order. Also interesting is to observe that $(id_{c^+} \oplus t_0) \oplus ((id_{c^+} \oplus t_1);t_2) = ((id_{c^+} \oplus t_0);t_2) \oplus (id_{c^+} \oplus t_1)$, i.e., we have no causal information about the token consumed by t_2: is it the one read by t_0, or the one read by t_1?

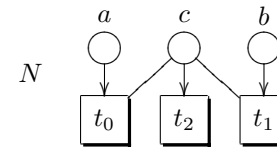


Fig. 5.

3 Individual Contexts

The maximum sharing hypothesis creates obvious problems when dealing with the *ITph*, whose entire point is to be able to recognize how electrons are emitted from tokens. For ordinary PT nets, the information about causality is recovered in the algebraic setting by using (non strictly) symmetric strict monoidal categories, i.e., by introducing *symmetries* for controlling rearrangements of tokens when composing processes. At the level of states we still have standard markings. At the level of computations (arrows), however, the tensor product is *not* commutative anymore, so that we are able to interpret correctly the flow of causality through token histories. Thus, the first attempt to a uniform extension of the *CTph* treatment of the previous section is to introduce symmetries on molecules.

There is however another problem to solve. Since the context $\varsigma(t)$ is modeled by a self-loop on $\varsigma(t)^{-}$, two transitions with the same context can be concatenated on it, as if one depended on the execution of the other. This spurious causal dependency is to be avoided, as it gives rise to a wrong semantic model. We thus choose to introduce a new kind of electrons, denoted by u_{-} for representing used (i.e., read) contexts. A transition t consume a *forward copy* of its context $\varsigma(t)^{-}$ and produces a *backward copy* $\varsigma(t)_{-}$ that cannot be read by other transitions. We call *bimolecules* the (generalized) markings of the algebra that includes also the operator $(_)_{-}$ subject to a set of axioms formally identical to those involving $(_)^{-}$ in Figure 2. Given a set S, we write $\nu(S)$ for the set of bimolecules on S.

The final and key ingredient in our construction is to *abandon* the symmetry of the monoidal categories involved. In a step similar to the one that brought from strictly symmetric to symmetric categories, we choose (*non* symmetric) monoidal categories to which we adjoin *exactly* and *only* the symmetries we need. In this way, we are able to omit those symmetries that would cause migration of electrons from atom to atom. In the following we shall build on the construction $\mathcal{Q}(N)$ for PT nets and, therefore, take a non commutative monoid of objects. We use the symbol $\otimes$ for the monoidal operation, which essentially amounts to string concatenation. Given a string q, we denote by $\mu(q)$ its underlying multiset.

Definition 7. *Given a contextual net $N = (S, T, \partial_0, \partial_1, \varsigma)$, we define the category $\mathcal{B}(N)$ as the category with objects the bimolecules on S and with arrows generated from the rules in Figure 6, together with the symmetries $\gamma_{a^\chi, b^\kappa} \colon a^\chi \otimes b^\kappa \to b^\kappa \otimes a^\chi$ and $\gamma_{a^\delta, c^\epsilon} \colon a^\delta \otimes c^\epsilon \to c^\epsilon \otimes a^\delta$, for $a, b, c \in S$ with $a \neq b$, for $\chi, \kappa \in \mathbb{N} \cup \{^{-}, _{-}\}$, and for $\delta, \epsilon \in \{^{-}, _{-}\}$. The arrows are taken modulo the axioms of strict monoidal categories in Figure 7 (whenever the γ's are defined) and the laws:*

$$s; t_{p,q}; s' = t_{p',q'} \tag{9}$$

$$\gamma_{a^\delta, a^\delta} = id_{a^\delta \otimes a^\delta}, \ \textit{for } \delta \in \{^{-}, _{-}\} \tag{10}$$

for any symmetries $s \colon p' \to p$ and $s' \colon q \to q'$, and any transition $t \colon \mu(p) \to \mu(q)$.

Note that we do *not* introduce the symmetries $\gamma_{a^k, a^{-}}$ and $\gamma_{a^k, a_{-}}$ that would allow the electrons to flow from a nucleus to a different one. For example, starting from $a \otimes a = a^{+} \otimes a_{-} \otimes a^{+} \otimes a^{-}$ and applying the arrow $a^{+} \otimes \gamma_{a_{-}, a^{+}} \otimes a^{-}$, we would reach

$$\frac{p \in \nu(S)}{id_p \colon p \to p} \qquad \frac{t \in T,\ \partial_0(t) \oplus \varsigma(t)^{\text{-}} = \mu(p),\ \partial_1(t) \oplus \varsigma(t)_{\text{-}} = \mu(q)}{t_{p,q} \colon p \to q} \qquad \frac{p, q \in S^{\otimes}}{\gamma_{p,q} \colon p \otimes q \to q \otimes p}$$

$$\frac{\alpha \colon p \to q,\ \beta \colon q \to r}{\alpha;\beta \colon p \to r} \qquad \frac{\alpha \colon p \to q,\ \beta \colon p' \to q'}{\alpha \otimes \beta \colon p \otimes p' \to q \otimes q'}$$

Fig. 6.

$$\begin{array}{ccc}
\alpha;(\beta;\sigma)=(\alpha;\beta);\sigma & \alpha;id_q=id_p;\alpha=\alpha & (\alpha;\beta)\otimes(\alpha';\beta')=(\alpha\otimes\alpha');(\beta\otimes\beta') \\
\alpha\otimes(\beta\otimes\sigma)=(\alpha\otimes\beta)\otimes\sigma & \alpha\otimes id_{\varnothing}=id_{\varnothing}\otimes\alpha=\alpha & id_{p\otimes q}=id_p\otimes id_q \\
(\alpha\otimes\beta);\gamma_{q,q'}=\gamma_{p,p'};(\beta\otimes\alpha) & \gamma_{p,q};\gamma_{q,p}=id_p\otimes id_q & \gamma_{p,q\otimes r}=(\gamma_{p,q}\otimes id_r);(id_q\otimes\gamma_{p,r})
\end{array}$$

Fig. 7.

$a^{\text{+}} \otimes a^{\text{+}} \otimes a_{\text{-}} \otimes a^{\text{-}} = a^{\text{+}} \otimes a \otimes a^{\text{-}}$, which is problematic. In fact, our *representation invariant* is that the electrons associated to a certain nucleus a^k in a string q are the first k electrons (either $a^{\text{-}}$ or $a_{\text{-}}$) that appear in q to the right of a^k. Thus, for consistency, we want exactly k electrons between a^k and the successive nucleus a^n occurring in q. The absence of those symmetries maintains this invariant.

As for $\mathcal{Q}(N)$ in [18], we introduce an arrow $t_{p,q}$ for all the possible linearizations p and q of the source and target of each transition t of N. Law (9), considered originally in [18], establishes a link between all the instances of a single t, guaranteeing both consistency and a sensible computational interpretation for such arrows. Actually, (9) expresses that the collection of the instances of t forms a *natural transformation* between suitable functors. The reader is referred to [18] for a thorough discussion of this topic. Laws (10) make the instances of electrons associated to the same nucleus indistinguishable from each other by asserting that the order in they are used is immaterial.

To establish our representation result we need to refine contextual processes in order to be able to concatenate them. As for similar cases in the literature, this leads to introduce an *ordering* of the tokens in the source and target of the process net, yielding the notion of *strongly concatenable contextual processes.*

Definition 8. *Given a net N, a* strongly concatenable contextual process *is a tuple $(\pi, \Theta, \prec_0, \prec_1)$, where π is a contextual process with underlying contextual process net Θ, $\prec_0$ and $\prec_1$ are total orders on the minimal and maximal places of Θ, respectively, such that $a \prec_0 b$ (resp. $a \prec_1 b$) implies $\pi_S(a) = \pi_S(b)$.*

Likewise concatenable processes, a partial operation of *sequential composition* can be defined. Provided the target of process π coincides with the source of process π', it merges the *maximal* places of π with the *minimal* places of π' according to the orders $\prec_1$ and $\prec_0'$. The *parallel composition* of two processes consists of taking their disjoint union and extending the orders on minimal and maximal places by taking $a \prec_i b$ whenever a belongs to the first process and b to the second. It can be shown that with these two operations the strongly

concatenable contextual processes of N form the arrows of a *strict monoidal category* $\mathcal{SCCP}(N)$. *Symmetries* can be defined by taking a process that contains just places (no transitions) with suitable orderings $\prec_0$ and $\prec_1$. Each place is both minimal and maximal. These symmetries make $\mathcal{SCCP}(N)$ be a symmetric monoidal category. Due to space limitation we cannot give more details here. We refer to [18] for the presentation of the category of strongly concatenable processes which is similar. We can now state the main result of the paper.

Theorem 2. *The category $\mathcal{SCCP}(N)$ is isomorphic (via a symmetric monoidal functor) to the full subcategory of $\mathcal{B}(N)$ whose objects are the elements of $S^{\otimes}$.*

The proof is quite long and requires the introduction and the description of the algebra of further kinds of processes that represent those arrows whose sources and targets involve nuclei and electrons. In particular, we use suitable inscriptions inside minimal and maximal places in order to represent the electrons which have been moved away from the nuclei of their atoms and their movements. However, such inscriptions are vacuous for processes whose source and target are strings of places (as all the electrons are next to their nuclei), and therefore we resort to strongly concatenable contextual process as in Theorem 2.

For example, let us consider again the net N in Figure 5. In $\mathcal{B}(N)$ we find the basic arrows $t_0' \colon a \otimes c^- \to c_-$, $t_0'' \colon c^- \otimes a \to c_-$, $t_1' \colon b \otimes c^- \to c_-$, $t_1'' \colon c^- \otimes b \to c_-$ and $t_2' \colon c \to \varnothing$, with $t_0' = \gamma_{a,c^-}; t_0''$ and $t_1' = \gamma_{b,c^-}; t_1''$. The concurrent execution of t_0 and t_1 with shared context can be written as $(id_{c^+} \otimes \gamma_{c^-,a} \otimes id_b); (id_{c^2} \otimes t_0'' \otimes t_1'')$. This time it differs from $t_0' \otimes id_{c^2} \otimes t_1''$ which, having source and target not in $S_N^{\otimes}$, does not correspond to any process. The concurrent execution of t_0 and t_1 with different contexts can be instead written as $\alpha = id_{c^+} \otimes t_0'' \otimes id_{c^+} \otimes t_1''$, and the terms $\alpha; (id_c \otimes t_2')$ and $\alpha; (t_2' \otimes id_c)$ denote different processes: in the former t_1 causes t_2 and in the latter t_0 causes t_2.

Besides the fact that *all* the arrows of $\mathcal{B}(N)$ have a meaningful computational interpretation, a further advantage of the present approach with respect to the match-share categories of [6] is that the arrows of the model category corresponding to pure concatenable process can be distinguished *just* by looking at their sources and targets, rather than by inspecting their construction.

Concluding Remarks and Future Work

Building on a illuminating suggestion of Meseguer in [9], we have shown a way to extend the algebraic semantics of PT nets proposed in [10] to contextual nets, both in the collective token and the individual token interpretation. The constructions rely on the choice of a non-free monoid of objects, whose elements we called molecules and bimolecules. Furthermore, in treating the individual token philosophy, we have renounced to the symmetry of the monoidal category, being then able to select only the symmetries consistent with our computational interpretation in terms of strongly concatenable contextual processes.

Although we have worked only at the level of single nets, we believe that our approach can be extended to constructions between categories of nets and models, with restrictions analogous to those well-known in the literature [17,18].

Acknowledgements. Thanks to José Meseguer and the referees for helpful suggestions.

References

1. P. Baldan, A. Corradini, and U. Montanari. An event structure semantics for P/T contextual nets: Asymmetric event structures. In *Proc. FoSSaCS'98*, vol. 1378 of *Lect. Notes in Comput. Sci.*, pp. 63–80. Springer, 1998.
2. E. Best and R. Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoretical Computer Science*, 55:87–136, 1987.
3. R. Bruni, J. Meseguer, U. Montanari, and V. Sassone. A comparison of Petri net semantics under the collective token philosophy. In *Proc. ASIAN'98*, vol. 1538 of *Lect. Notes in Comput. Sci.*, pp. 225–244. Springer, 1998.
4. S. Christensen and N.D. Hansen. Coloured Petri nets extended with place capacities, test arcs and inhibitor arcs. In *Applications and Theory of Petri Nets*, vol. 691 of *Lect. Notes in Comput. Sci.*, pp. 186–205. Springer, 1993.
5. P. Degano, J. Meseguer, and U. Montanari. Axiomatizing the algebra of net computations and processes. *Acta Inform.*, 33(7):641–667, 1996.
6. F. Gadducci and U. Montanari. Axioms for contextual net processes. In *Proc. ICALP'98*, vol. 1443 of *Lect. Notes in Comput. Sci.*, pp. 296–308. Springer, 1998.
7. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. *Inform. and Comput.*, 57:125–147, 1983.
8. R. Janicki and M. Koutny. Semantics of inhibitor nets. *Inform. and Comput.*, 123:1–16, 1995.
9. J. Meseguer. Rewriting logic as a semantic framework for concurrency: A progress report. In *Proc. CONCUR'96*, vol. 1119 of *LNCS*, pp. 331–372. Springer, 1996.
10. J. Meseguer and U. Montanari. Petri nets are monoids. *Inform. and Comput.*, 88(2):105–155, 1990.
11. J. Meseguer, U. Montanari, and V. Sassone. On the semantics of place/transition Petri nets. *Mathematical Structures in Computer Science*, 7:359–397, 1997.
12. U. Montanari and F. Rossi. Contextual occurrence nets and concurrent constraint programming. In *Graph Transformations in Computer Science*, vol. 776 of *Lect. Notes in Comput. Sci.*, pp. 280–285. Springer, 1994.
13. U. Montanari and F. Rossi. Contextual nets. *Acta Inform.*, 32:545–596, 1995.
14. C.A. Petri. *Kommunikation mit Automaten.* Ph.D. thesis, Institut für Instrumentelle Mathematik, Bonn, 1962.
15. W. Reisig. *Petri Nets: An Introduction.* EACTS Monographs on Theoretical Computer Science. Springer, 1985.
16. G. Ristori. *Modelling Systems with Shared Resources via Petri Nets.* Ph.D. thesis, Dipartimento di Informatica, Università di Pisa, 1994.
17. V. Sassone. An axiomatization of the algebra of Petri net concatenable processes. *Theoretical Computer Science*, 170:277–296, 1996.
18. V. Sassone. An axiomatization of the category of Petri net computations. *Mathematical Structures in Computer Science*, 8:117–151, 1998.
19. R.J. van Glabbeek and G.D. Plotkin. Configuration structures. In *Proc. LICS'95*, pp. 199–209. IEEE Press, 1995.
20. W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In *Proc. ICALP'97*, vol. 1256 of *Lect. Notes in Comput. Sci.*, pp. 538–548. Springer, 1997.
21. W. Vogler. Partial order semantics and read arcs. In *Proc. MFCS'97*, vol. 1295 of *Lect. Notes in Comput. Sci.*, pp. 508–517. Springer, 1997.
22. G. Winskel. Event structures. In *Proc. of Advanced Course on Petri Nets*, vol. 255 of *Lect. Notes in Comput. Sci.*, pp. 325–392. Springer, 1986.

Asymptotically Optimal Bounds for OBDDs and the Solution of Some Basic OBDD Problems
(Extended Abstract)

Beate Bollig* and Ingo Wegener*

FB Informatik, LS2, Univ. Dortmund, 44221 Dortmund, Germany
bollig,wegener@ls2.cs.uni-dortmund.de

Abstract. Ordered binary decision diagrams (OBDDs) are nowadays the most common dynamic data structure or representation type for Boolean functions. Among the many areas of application are verification, model checking, and computer aided design. For many functions it is easy to estimate the OBDD size but asymptotically optimal bounds are only known in simple situations. In this paper, methods for proving asymptotically optimal bounds are presented and applied to the solution of some basic problems concerning OBDDs. The largest size increase by a synthesis step of π-OBDDs followed by an optimal reordering is determined as well as the largest ratio of the size of deterministic finite automata and quasi-reduced OBDDs compared to the size of OBDDs. Moreover, the worst case OBDD size of functions with a given number of 1-inputs is investigated.

1 Introduction and Results

Branching programs (BPs) are a well established representation type or computation model for Boolean functions. Its size is tightly related to the nonuniform space complexity (see e.g. [14]). Hence, one is interested in exponential lower bounds for more and more general types of BPs (for the latest breakthrough for semantic linear depth BPs see [1]). In order to use variants of BPs as dynamic data structure one needs a BP variant such that a list of important operations (see e.g. [15]) can be performed efficiently. E.g., for verification, model checking, and a lot of CAD applications we need an efficient test whether a representation has a satisfying input (satisfiability test) and an efficient test whether two representations describe the same function (equality test). These are NP-hard problems for general BPs.

Bryant [4] has presented π-OBDDs as a simple BP variant allowing efficient algorithms for all important operations. Although we now have efficient algorithms for more general and, therefore, more compact representation types, π-OBDDs are used in most applications and the use of an OBDD package [13] is nowadays a standard technique.

* Supported in part by DFG grant We 1066/8.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 187–198, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Definition 1. *Let $X_n = \{x_1, \ldots, x_n\}$ be a set of Boolean variables. A variable ordering π on X_n is a permutation on $\{1, \ldots, n\}$ leading to the ordered list $x_{\pi(1)}, \ldots, x_{\pi(n)}$ of the variables.*

Definition 2. *A π-OBDD on X_n is a directed acyclic graph $G = (V, E)$ whose sinks are labeled by Boolean constants and whose non sink (or inner) nodes are labeled by Boolean variables from X_n. Each inner node has two outgoing edges one labeled by 0 and the other by 1. The edges between inner nodes have to respect the variable ordering π, i.e., if an edge leads from an x_i-node to an x_j-node, $\pi^{-1}(i) \leq \pi^{-1}(j)$ (x_i precedes x_j in $x_{\pi(1)}, \ldots, x_{\pi(n)}$). Each node v represents a Boolean function $f_v : \{0,1\}^n \to \{0,1\}$ defined in the following way. In order to evaluate $f_v(a)$, $a \in \{0,1\}^n$, start at v. After reaching an x_i-node choose the outgoing edge with label a_i until a sink is reached. The label of this sink defines $f_v(a)$. The size of the π-OBDD G is equal to the number of its nodes.*

Bryant [4] has already shown that the minimal-size π-OBDD for a function f is unique (up to isomorphism) and it is called the reduced π-OBDD (or shortly the π-OBDD) for f. Its size is described by the following structure theorem [12]. In order to simplify the description we describe the theorem only for the special case where π equals the identity $id(i) = i$.

Theorem 1. *The number of x_i-nodes of the id-OBDD for f is the number s_i of different subfunctions $f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$, $a_1, \ldots, a_{i-1} \in \{0,1\}$, essentially depending on x_i (a function g depends essentially on x_i if $g_{|x_i=0} \neq g_{|x_i=1}$).*

It is a simple corollary that the number s_i^* of different subfunctions $f_{|x_1=a_1,\ldots,x_{i-1}=a_{i-1}}$, $a_1, \ldots, a_{i-1} \in \{0,1\}$, is a lower bound on the *id*-OBDD size of f. Obviously, $\lceil \log s_i^* \rceil$ is the one-way deterministic communication complexity of f if Alice holds $x_1, \ldots, x_{i-1}$ and Bob holds $x_i, \ldots, x_n$. (See [5] and [7] for the theory of communication complexity.) For non-constant functions the *id*-OBDD size of f equals $s_1 + \cdots + s_n + 2$. If many s_i have asymptotically the same and the largest value, one-way communication complexity is not strong enough to obtain asymptotically optimal bounds. Moreover, we have the freedom to choose an appropriate variable ordering for f. Let π-OBDD(f) denote the π-OBDD size of f.

Definition 3. *The OBDD size of f (denoted by OBDD(f)) is the minimum of all π-OBDD(f).*

Using Theorem 1 or one-way communication complexity a lot of exponential lower bounds on the OBDD size of functions have been proved (see e.g. [15]). But there are only a few functions whose OBDD size is asymptotically known exactly. These are functions with linear OBDD size, symmetric functions, and a few more functions. We develop lower bound methods for asymptotically optimal OBDD bounds and solve problems motivated from OBDD applications, automata theory, and complexity theory.

In order to use OBDDs we have to transform a logical description of a function, e.g. a circuit, into an OBDD representation. This is done by a sequence

of binary synthesis steps. A binary synthesis step computes a π-OBDD G_h for $h = f \otimes g$ ($\otimes$ is a binary Boolean operation like AND or EXOR) from π-OBDDs G_f and G_g for f resp. g. Bryant [4] has shown how this can be done in time $O(|G_f| \cdot |G_g|)$ and he has presented an example that the result may need size $\Theta(|G_f| \cdot |G_g|)$. His example has two drawbacks. The chosen variable ordering is bad for h, f, and g (and, therefore, such a synthesis step will not occur in applications) and the functions f and g depend essentially on disjoint sets of variables. It is not too hard to present an example without these drawbacks. But this is nevertheless not the answer to the question about the worst case for the binary synthesis problem. If a binary step leads to a π-OBDD much larger than the given π-OBDDs, all recent OBDD packages start to look for a better variable ordering. Although the search for an optimal OBDD variable ordering is NP-hard [2] and this holds even for the corresponding approximation problems for arbitrary constant factors [11], heuristic algorithms like sifting [10] often lead to very good results. Hence, the main step is a binary synthesis step followed by reordering. This leads to the problem whether it is possible that $\text{OBDD}(h) = \Theta(\pi\text{-OBDD}(f) \cdot \pi\text{-OBDD}(g))$ for functions f and g essentially depending on all considered variables. (Here we should mention the folklore result (see [15]) that $\text{OBDD}(h)$ may be exponential even if $\text{OBDD}(f)$ and $\text{OBDD}(g)$ are linear but the linear size only is possible for different variable orderings.) In Section 4, we solve the problem by representing an example where $\pi\text{-OBDD}(f_n) = \Theta(n)$, $\pi\text{-OBDD}(g_n) = \Theta(n)$, and $\text{OBDD}(h_n) = \Theta(n^2)$. This surely is the less surprising answer but the lower bound proof for the OBDD size of h_n has some interesting features.

Some applications [9] work with a restricted variant of π-OBDDs which may be called leveled π-OBDDs or quasi-π-OBDDs (π-QOBDDs).

Definition 4. *A π-QOBDD is a π-OBDD with the additional property that each edge leaving an $x_{\pi(i)}$-node, $i < n$, reaches an $x_{\pi(i+1)}$-node.*

We are interested in QOBDDs also because of their tight relationship to deterministic finite automata (DFAs) for so-called Boolean languages L where $L \subseteq \{0,1\}^n$ for some n. It is an easy exercise to verify for $L_f = f^{-1}(1)$ that $\text{DFA}(L_f) \leq$ *id*-$\text{QOBDD}(f) \leq \text{DFA}(L_f) + n$. Hence, *id*-QOBDDs and DFAs are almost the same. For general regular languages consisting of strings of different lengths it makes no sense to discuss different "variable orderings" or permutations of the input string. For Boolean languages, a π-DFA may apply the reordering π to all inputs of length n. The above inequality can be generalized to $\pi\text{-DFA}(L_f) \leq \pi\text{-QOBDD}(f) \leq \pi\text{-DFA}(L_f) + n$. Moreover, it is obvious that $\pi\text{-QOBDD}(f) \leq (n+1) \cdot \pi\text{-OBDD}(f)$ and this bound is tight for the constant functions (syntactically depending on n variables). It is also not too difficult to see that $\pi\text{-QOBDD}(f) = \Theta(n \cdot \pi\text{-OBDD}(f))$ for some function f essentially depending on all n variables.

Definition 5. *The multiplexer MUX_n (or direct storage access function DSA_n) is defined on $n + k$ variables $a_{k-1}, \ldots, a_0, x_0, \ldots, x_{n-1}$ where $n = 2^k$.*

$MUX_n(a,x) = x_{|a|}$ where $|a|$ is the number whose binary representation equals $(a_{k-1},\ldots,a_0)$.

Let π be the variable ordering $(a_{k-1},\ldots,a_0,x_0,\ldots,x_{n-1})$. Then π-OBDD(MUX$_n$) = OBDD(MUX$_n$) $= 2n+1$. The π-OBDD starts with a complete binary tree on the a-variables. For the path where $|a| = i$ it is sufficient to test x_i. For the π-OBDD we need i extra nodes before the x_i-node and $n-1$ extra nodes before each of the sinks. Hence, π-QOBDD(MUX$_n$) $= \frac{1}{2}n^2 + \frac{7}{2}n - 1 = \Theta(n \cdot \pi\text{-OBDD(MUX}_n))$. But in order to compare the size of OBDDs and QOBDDs (and DFAs for Boolean languages with reordering) we ask whether QOBDD$(f_n) = \Theta(n \cdot \text{OBDD}(f_n))$ for functions f_n essentially depending on n variables. This is the question whether the possibility of OBDDs to omit the test of variables may save a size factor of $\Theta(n)$.

Since it is the main rule of thumb for the variable ordering problem to test control or address variables (like the a-variables of MUX$_n$) before the data variables (like the x-variables of MUX$_n$), it was a well-established conjecture that the considered variable ordering π is optimal for QOBDDs for MUX$_n$ and that QOBDD(MUX$_n$) $= \Theta(n^2)$. In Section 2, we prove the surprising result that QOBDD(MUX$_n$) $= \Theta(n^2/\log n)$. Section 3 provides a function f_n essentially depending on n variables such that QOBDD$(f_n) = \Theta(n \cdot \text{OBDD}(f_n))$ proving that the freedom of OBDDs to omit tests may indeed decrease the size by a factor of $\Theta(n)$.

In Section 5, we investigate the dependence of the OBDD size on the size of $f_n^{-1}(1)$. Let $N(a(n))$ be the number of Boolean functions f where $|f_n^{-1}(1)| \le a(n)$. The standard counting argument proves the existence of functions f_n where $|f_n^{-1}(1)| \le a(n)$ such that its OBDD size and even its circuit size is $\Omega\big(\log N(a(n))/\log\log N(a(n))\big)$. On the other hand, obviously OBDD$(f_n) \le$ O$(na(n))$ for these functions. For $a(n) = 2^n$, the lower bound of size $2^n/n$ is optimal (see [15]). For $a(n) = 1$, the upper bound of size n is optimal. The question is how large can we choose $a(n)$ such that we can define functions f_n where $|f_n^{-1}(1)| \le a(n)$ and OBDD$(f_n) = \Theta(n\,a(n))$. We describe such functions for polynomially increasing $a(n)$.

2 QOBDDs and DFAs for the Multiplexer

In this section, we determine the size of QOBDDs and DFAs with reordering for the representation of the multiplexer.

Theorem 2. *QOBDD$(MUX_n) = \Theta(n^2/\log n)$.*

Sketch of proof. First, we prove for some variable ordering π that the π-QOBDD size of MUX$_n$ is $O(n^2/\log n)$. Let $m := k - \lfloor \log k \rfloor + 1$. The variable ordering π is given by $a_{k-1},\ldots,a_{k-m},x_0,\ldots,x_{n-1},a_{k-m-1},\ldots,a_0$. The x-variables are partitioned to 2^m groups such that the indices of the variables of each group agree in their binary representation in the m most significant bits. The size of each group is $n/2^m$. We start with a complete bi-

nary tree of the first m a-variables. The tree has 2^m leaves and $2^m - 1$ inner nodes. Then we test the x-variables of group G_0. Only the subfunction where $a_{k-1} = \cdots = a_{k-m} = 0$ essentially depends on these variables. For all other groups we need x_i-nodes, since we consider QOBDDs. Hence, we need one complete binary tree with $2^{n/2^m}$ leaves and $2^{n/2^m} - 1$ inner nodes and $(2^m - 1)n/2^m < n$ further nodes which could be eliminated in OBDDs. One leaf can be replaced by the 0-sink. The same arguments work for the next group. Here the width for the first group is $2^{n/2^m}$ and the total width is bounded by $2 \cdot 2^{n/2^m} + 2^m - 2$. The crucial argument is the following one. We can merge the $2^{n/2^m}$ nodes for the case $(a_{k-1}, \ldots, a_{k-m}) = (0, \ldots, 0, 0)$ with the $2^{n/2^m}$ nodes for the case $(a_{k-1}, \ldots, a_{k-m}) = (0, \ldots, 0, 1)$. We only have to store the data vector namely the x-vector. The result only depends on the further address bits. The width after the tests of the x-variables of group G_{2^m-1} equals $2^{n/2^m} - 1$. The size of the last $k - m$ levels is bounded by $2^{2^{k-m}}$. The total size is bounded above by $2^m - 1 + 2^{2^{k-m}} + n(2 \cdot 2^{n/2^m} + 2^m - 2) \leq n2^m + 2n2^{n/2^m} + 2^{2^{k-m}}$. By the choice of m, $2n/\log n \leq 2^m \leq 4n/\log n$, and we obtain an upper bound of $O(n^2/\log n)$. For the lower bound for arbitrary variable orderings π it is sufficient to prove a lower bound of size $\Omega(n/\log n)$ on the size of $\Omega(n)$ x-levels of π-QOBDDs. By standard lower bound techniques it can be shown that the x_i-level of a π-QOBDD representing MUX_n has a size of $\Omega(n/\log n)$ if x_i belongs to the second quarter of all x-variables with respect to π (for more details see [3]). □

This result implies by the discussion in Section 1 the same result for DFAs with reordering. We only want to mention here that we can get similar results for so-called zero-supressed BDDs (ZBDDs) which are used in many applications (see e.g. [8]). This is the first example of a function (moreover, a "natural" function) and of BDD models of practical relevance that the rule of thumb "control variables before data variables" is wrong.

3 The Maximal Size Gap between OBDDs, QOBDDs and DFAs

We look for functions f_N essentially depending on all their N variables such that the size gap of OBDDs on one hand and QOBDDs and DFAs with reordering on the other hand is a factor of $\Theta(N)$ which is by the discussion in Section 1 the largest possible gap. For such a function it is necessary that many edges in the optimal OBDD omit many tests. Therefore, the multiplexer has been considered as a good candidate. But the result of Section 2 implies that the multiplexer only leads to a gap of $\Theta(N/\log N)$. The multiplexer has many more data variables than address variables. We can prove the largest possible gap for a function f_N on $N = n^2 + 2n$ variables, among them n^2 control or address variables (here called selection variables) $s_0, \ldots, s_{n^2-1}$ and only $2n$ data variables $x_0, y_0, \ldots, x_{n-1}, y_{n-1}$ which lead to n^2 data pairs $x_i y_j$, $0 \leq i, j \leq n-1$. The data pairs are partitioned to n blocks B_m, $0 \leq m \leq n-1$, such that B_m contains

all pairs $x_i y_{i+m}$ (the indices are computed mod n). We consider the following ordering $p_0, \ldots, p_{n^2-1}$ of the pairs. The pair p_k where $k = in + j$ equals $x_j y_{i+j}$, i.e., we start with the pairs from B_0, followed by the pairs from B_1, ... In each block we start with the pair containing x_0, followed by the pair containing x_1, and so on. The main property is that the distance between two pairs containing x_i equals n and the distance between two pairs containing y_i is at least $n-1$. Finally, we define f_N by $f_N(s,x,y) = \bigvee_{0 \le k \le n^2-1} s_0 \ldots s_{k-1} \overline{s_k} p_k$, i.e., the s-vector selects with its first zero which pair has to be evaluated.

Theorem 3. *OBDD*$(f_N) = \Theta(N)$ *while QOBDD*$(f_N) = \Theta(N^2)$ *and DFA*$(f_N) = \Theta(N^2)$.

Sketch of proof. It is obvious that the OBDD size of f_N for the "natural" variable ordering $s_0, \ldots, s_{n^2-1}, x_0, \ldots, x_{n-1}, y_0, \ldots, y_{n-1}$ equals $2n^2+n+2 = \Theta(N)$ and that f_N essentially depends on all its variables. This implies OBDD$(f_N) = \Theta(N)$ and the upper bounds for the other models.

In the following we present a sketch of the proof for the lower bound on the QOBDD size (for more details see [3]). This implies the lower bound for DFAs with reordering. We fix an arbitrary variable ordering π. There are $n^2/24$ levels where a selection variable is tested and the number of already tested selection variables is at least $n^2/8$ and less than $n^2/6$. It is sufficient to prove that each of these levels has a size of $\Omega(n^2)$.

We fix one of the described levels and use the following notation. The sets $T(x)$, $T(y)$, and $T(s)$ contain the x-, y-, and s-variables, resp., which are tested before the considered levels. The sets $R(x)$, $R(y)$, and $R(s)$ contain the corresponding remaining variables. Let $T(x,y) := T(x) \cup T(y)$. We distinguish whether the size of $T(x,y)$ is small, large, or medium.

Case 1: $|T(x,y)| \le \frac{1}{9}n$ (small size).

Case 2: There are at least $2 \log n$ variables in $T(x)$ (or $T(y)$) such that for each of these variables x_i there is a pair $p_k = x_i y_j$ where $s_k \in R(s)$ (large size).

Remark: This case is called "large size", since one of the conditions $|T(x)| \ge \frac{1}{6}n + 2\log n$ and $|T(y)| \ge \frac{1}{6}n + 2\log n$ is sufficient for Case 2.

Case 3: Not Case 1 or Case 2 (medium size).

If $|T(x,y)|$ is small, we can argue similarly to the case of the natural variable ordering. If it is large, we have to store too much information on the data variables like in the case of the multiplexer. The most difficult case is the case where $T(x,y)$ is of medium size. We assume that $|T(x)| \ge \frac{1}{18}n$ (the other case can be handled similarly). There is a subset $T'(x)$ of $T(x)$ of size $\frac{1}{18}n - 2\log n \ge \frac{1}{20}n$ (for large n) such that $s_k \in T(s)$ for all pairs $p_k = x_i y_j$ where $x_i \in T'(x)$. The condition $|T(y)| < \frac{1}{6}n + 2\log n \le \frac{1}{5}n$ (for large n) implies for each $x_i \in T'(x)$ the existence of at least $\frac{4}{5}n$ pairs $p_k = x_i y_j$ such that $s_k \in T(s)$ and $y_j \in R(y)$.

We have partitioned the set of all pairs $p_0, \ldots, p_{n^2-1}$ into n blocks $B_0, \ldots, B_{n-1}$ such that $B_i = \{p_{in}, \ldots, p_{in+n-1}\}$.

Claim 1. *There are* $\frac{7}{9}n$ *good blocks, i.e., blocks each containing at least* $\frac{1}{200}n$ *of the chosen pairs.*

Now we investigate the set P of the $\Omega(n^2)$ chosen pairs p_k belonging to the $\frac{7}{9}n$ good blocks. For each such pair $p_k = x_i y_j$ we know that $s_k \in T(s)$, $x_i \in T(x)$, and $y_j \in R(y)$. Moreover, we define a subfunction g_k of f_N by assigning the following values to the tested variables. We assign the value 0 to s_k and the value 1 to all other variables in $T(s)$. We assign the value 1 to x_i and the value 0 to all other variables in $T(x, y)$. The function g_k has a prime implicant consisting of y_j and perhaps some s-variables. It is possible that $g_k = g_l$, $k \neq l$, but then we have some implications on the set $T(s)$.

Claim 2. *If $k < l$ and $g_k = g_l$, then $s_{k+1}, \ldots, s_{l-1} \in T(s)$ and the pairs p_k and p_l contain the same y-variable.*

Since each variable is contained in exactly one pair in each block, $g_k = g_l$ implies that p_k and p_l belong to different blocks.

Claim 3. *There are less than $\frac{1}{2}n$ blocks containing some $p_k \in P$ such that $g_k = g_l$ for some $p_l \in P$ and $l \neq k$.*

¿From Claim 1 and Claim 3 we conclude that there are at least $(\frac{7}{9} - \frac{1}{2})n = \frac{5}{18}n$ blocks each containing $\frac{1}{200}n$ pairs such that all the corresponding $\Omega(n^2)$ subfunctions g_k are different. This implies the lower bound $\Omega(n^2)$ on the size of the level. □

With similar arguments we can prove that $\mathrm{ZBDD}(f_N) = \Theta(N^2)$.

4 The Maximal Size Increase of a π-OBDD Synthesis Step with Optimal Reordering

In this section, we prove that the synthesis of π-OBDDs essentially depending on the same set of variables can lead to a multiplicative size increase (like the well-known result for DFAs) and that this result even holds if the synthesis can be followed by an optimal reordering. The functions are defined on $n + 2k$ variables $x_0, \ldots, x_{k-1}, y_0, \ldots, y_{k-1}, z_0, \ldots, z_{n-1}$ where $n = 2^k$. Let $f_n(x, y, z) = \mathrm{MUX}_n(x, z)$ and $g_n(x, y, z) = \mathrm{MUX}_n(y, z)$. These functions do not depend essentially on all variables. Nevertheless, we first investigate f_n, g_n, and $h_n = f_n \oplus g_n$. Afterwards, we define modified functions f_n^*, g_n^*, and $h_n^* = f_n^* \oplus g_n^*$ depending essentially on all variables and having similar properties. The function h_n is defined by

$$h_n(x, y, z) = \bigvee_{0 \leq i,j \leq n-1} (|x| = i) \wedge (|y| = j) \wedge (z_i \oplus z_j).$$

Theorem 4. *Let π^* be the variable ordering $x_0, \ldots, x_{k-1}, y_0, \ldots, y_{k-1}, z_0, \ldots, z_{n-1}$. Then π^*-OBDD$(f_n) = \pi^*$-OBDD$(g_n) = 2n + 1$ but OBDD$(h_n) = \Omega(n^2)$, i.e., a synthesis step followed by optimal reordering can lead to a multiplicative size increase.*

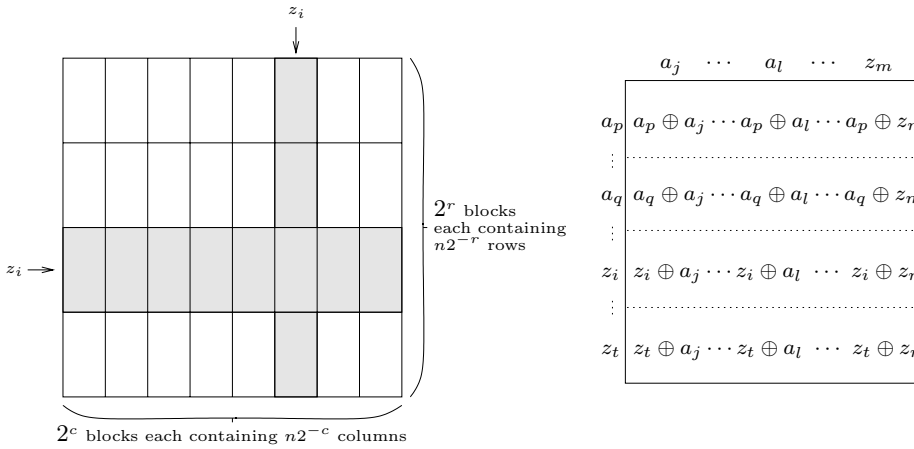


Fig. 1a) *A macroscopic view* **b)** *A microscopic view*

We prove a lower bound of size $\Omega(n^2)$ by proving that each of $\Omega(n)$ levels has size $\Omega(n)$. The interesting aspect is that there is not necessarily a block of $\Omega(n)$ levels each of size $\Omega(n)$. It may happen that small levels and large levels occur in a rather irregular order. Nevertheless, we are able to bound the number of small levels in a sufficient way. This is the first proof of an asymptotically optimal OBDD lower bound in such a situation.

Sketch of proof of Theorem 4. The upper bounds are obvious. For the lower bound proof we fix an arbitrary variable ordering π. First, we visualize the situation after the test of some variables. We do not use the communication matrix, since we believe that a different representation better supports the counting of different subfunctions essentially depending on some specific z-variable. Again we use the notation $T(x)$, $T(y)$, and $T(z)$ for the sets of already tested variables. Let $r := T(x)$ and $c := T(y)$. Then we have 2^r partial x-addresses which partition the set of all z-variables into 2^r blocks of size $n2^{-r}$ each. Two z-variables z_i and z_j belong to the same block if the binary representations of i and j agree in the positions belonging to variables in $T(x)$. In the same way we obtain 2^c blocks of size $n2^{-c}$ each corresponding to the variables in $T(y)$. We consider the following $n \times n$-matrix. The rows correspond to the z-variables and they are ordered blockwise with respect to the 2^r row blocks. In each block we order the variables according to the canonical ordering with respect to the vector describing the value of the x-variables which have not been tested yet. The columns also correspond to the z-variables and they are ordered blockwise with respect to the 2^c column blocks. The entry at the z_i-row and the z_j-column equals $z_i \oplus z_j$. Our aim is to prove a lower bound on the size of the z_i-level of the π-OBDD representing h_n. For this reason it is sufficient to investigate those $2^r + 2^c - 1$ assignments to the variables in $T(x, y)$ where at least one of the partial addresses allows the value i. In Fig. 1a) the corresponding blocks are shaded. Our aim is to

count the number of different subfunctions essentially depending on z_i. First, we consider the subfunctions for some fixed assignment to the variables in $T(x, y)$. This leads to a submatrix of the matrix considered above (see Fig. 1b)). In our example the submatrix contains the z_i-row but not the z_i-column. It is called a z_i-row-rectangle. Variables $z_j \in T(z)$ are replaced by a_j. This z_i-row-rectangle is a description of the considered subfunction of h_n. Let s be the number of different variables from $T(z)$ which correspond to a column or row of the considered submatrix. Then we obtain exactly 2^s different subfunctions all essentially depending on z_i. Similar results hold for z_i-column-rectangles.

Is it possible to obtain the same subfunction for different assignments to the variables from $T(x, y)$? This happens iff the a-entries are replaced in such a way by constants that two z_i-rectangles are equal. We assume that $r < k$ or $c < k$ (if $r = c = k$, all address variables have been tested which leads to an easy subcase). A z_i-row-rectangle R_i differs from a z_i-column-rectangle C_i, since R_i contains entries essentially depending on z_i exactly in the z_i-row while this happens in C_i exactly in the z_i-column. Now we consider w.l.o.g. two z_i-row-rectangles R_i and R_i'. If R_i contains a z_m-column, it contains a column where all entries essentially depend on z_m. This cannot happen in R_i' where at most one row can depend essentially on z_m. Hence, two different z_i-row-rectangles agree iff all z-variables corresponding to the columns have already been tested and the corresponding vectors are equal. The only-if-part follows from the consideration of the case $|x| = i$. The if-part follows, since the remaining assignments to x-variables and z-variables belonging to rows of the block have the same influence on R_i and R_i'.

Summarizing we can conclude that we are able to determine the size of each z-level of a π-OBDD representing h_n. We still have to prove that $\Omega(n)$ z-levels have size $\Omega(n)$. The first and last z-levels can be very small. We concentrate on the levels where the z-variables at the positions $n/2+1, \ldots, 3n/4$ of the variable ordering on the z-variables are tested.

Case 1. There is no block such that all corresponding row variables have already been tested. (The same arguments work for the column variables.)

We consider the z_i-column-rectangles. Let r_j be the number of $T(z)$-variables belonging to the jth row block, $1 \leq j \leq 2^r$. The sum of all r_j is at least $n/2$ (by the choice of the considered levels) and our lower bound arguments lead to the lower bound $\sum_{1 \leq j \leq 2^r} 2^{r_j}$ on the size of the considered z_i-level. This lower bound is minimal if the sum of all r_j is equal to $n/2$ and all r_j are equal to $\frac{n}{2}/2^r$. This leads to the lower bound $2^r 2^{n/2^{r+1}}$. As long as $n/2^{r+1} \geq 2$, this exponent decreases at least by 1 if r is increased by 1. In these cases the lower bound is decreasing with r. This happens as long as $r + 2 \leq \log n$. Hence, we obtain an $\Omega(n)$ bound on the size of the z_i-level.

Case 2. Not Case 1 and $r \leq \log n - \log\log n$ (or $c \leq \log n - \log\log n$).

There is a z_i-column-rectangle where all $n2^{-r}$ z-variables corresponding to the rows of the rectangle have already been tested. This leads to a lower bound of size $2^{n2^{-r}} \geq n$.

Case 3. $r \geq \log n - c^*$ (or $c \geq \log n - c^*$) for some constant c^* ($c^* = 8$ is appropriate in this proof).

We have 2^r z_i-column-rectangles and at least $n/4$ z-variables which have not been tested. Each row block contains $n2^{-r}$ z-variables. Hence, there are at least $\frac{1}{4}2^r = \Omega(n)$ z_i-column-rectangles such that at least one row variable has not been tested. For each of these rectangles we obtain a lower bound of size 1 and we have shown above that the different z_i-column-rectangles cannot agree and describe different subfunctions. Altogether, we obtain also in this case a lower bound of size $\Omega(n)$.

We obtain the proposed $\Omega(n^2)$ lower bound if $\Omega(n)$ of the $n/4$ considered levels belong to one of the three cases. Hence, we only have to consider the situation where $n/4 - o(n)$ of the considered levels do not fulfil the assumptions of one of the three cases. We assume w.l.o.g. that on $n/8 - o(n)$ of these levels the condition $c \leq r$ holds. On these levels, $\log n - \log\log n < c \leq r < \log n - c^*$. Only to simplify the notation we assume that $N = \log\log n$ is an integer. We consider the levels where $r = \log n - \log\log n + t$ has a fixed value. In particular, $1 \leq t \leq \log\log n - c^*$. We have $2^r = 2^t n/\log n$ z_i-column-rectangles. Since at least $n/4$ z-variables have not been tested, there are at least 2^{r-2} z_i-column-rectangles such that some corresponding row variable has not been tested. Let m be the number of already tested z-variables belonging to the column block containing z_i. This leads to the lower bound $\frac{1}{4}2^r 2^m$. If $m \geq \log\log n - t$, the lower bound is of size $\Omega(n)$. We have 2^c column blocks of size $n2^{-c}$ each. Hence, there are at most $2^c(\log\log n - t) \leq 2^r(\log\log n - t)$ bad z-levels where we have not proved an $\Omega(n)$ bound. Let $n \geq 4$. Then the number of bad levels can be estimated by $\sum_{1\leq t<N-c^*} 2^t \frac{n}{\log n}(N-t) \leq \frac{n}{\log n}(c^*+2)2^{-c^*}2^N = (c^*+2)2^{-c^*}n$. For $c^* = 8$ these are at most $\frac{10}{256}n$ bad levels out of $\frac{1}{8}n - o(n)$ levels. Hence, also in this situation we have proved the existence of $\Omega(n)$ levels whose size is $\Omega(n)$. This implies the proposed $\Omega(n^2)$ bound. □

In a last step, we have to generalize our results to functions essentially depending on all their variables (for details see [3]).

5 On the Maximal OBDD Size with Respect to the Number of 1-Inputs

For the construction of a function $f_{n,k}$ with $|f_n^{-1}(1)| = \binom{n}{k}$ and $\mathrm{OBDD}(f_{n,k}) = \Theta(n|f_n^{-1}(1)|)$ we use the construction of Kovari, Sós, and Turán [6] for the solution of the well-known problem of Zarankiewicz. Their result can be explained as follows. Let $n = p^2$ for some odd prime p. Let $U := \{0, \ldots, n-1\}$ be the universe which is partitioned to p blocks $B_0, \ldots, B_{p-1}$ where $B_i = \{ip, \ldots, ip+p-1\}$. Then it is possible to define explicitly sets $A_0, \ldots, A_{n-1}$ with the following properties:

- $|A_i| = p$ for all i,
- $|A_i \cap A_j| \leq 1$ for all $i \neq j$,
- $|A_i \cap B_j| = 1$ for all i and j,
- for all $i \in B_k$ and $j \in B_l$ where $k \neq l$ there exists some m such that $i, j \in A_m$.

For the definition of $f_{n,k}$ we consider for each choice of $0 \leq i_1 < i_2 < \cdots < i_k \leq n$

the set $A_{i_1,\ldots,i_k}$ defined as the union of all $A_{i_j}, 1 \leq j \leq k$, and the corresponding minterm $m_{i_1,\ldots,i_k}$ on $\{x_0,\ldots,x_{n-1}\}$ which computes 1 iff $x_i = 1$ exactly for all $i \in A_{i_1,\ldots,i_k}$. The function $f_{n,k}$ computes 1 iff one of the minterms $m_{i_1,\ldots,i_k}$ computes 1.

Theorem 5. *Let k be a constant. Then $|f_n^{-1}(1)| \leq \binom{n}{k}$ and $OBDD(f_{n,k}) = \Theta(n\binom{n}{k})$.*

Proof. By definition, $|f_{n,k}^{-1}(1)| \leq \binom{n}{k}$. (For large n, even $|f_{n,k}^{-1}(1)| = \binom{n}{k}$.) This implies the upper bound $n\binom{n}{k} + 2$ on the OBDD size of $f_{n,k}$, since at most $\binom{n}{k}$ of all assignments to some set of variables can be different from the constant 0.

For the lower bound proof we fix a variable ordering π and investigate the set P of all 1-paths namely all computation paths $p_{i_1,\ldots,i_k}$ corresponding to the minterms $m_{i_1,\ldots,i_k}$. The proof strategy is the following one. We identify a set $P' \subseteq P$ such that two different paths from P' have been split before or at level $\frac{1}{3}n$, i.e., there is a node where one path chooses the 0-edge and the other one chooses the 1-edge. Afterwards, we identify a subset of P' such that two paths from this subset cannot share a node on the levels $\frac{1}{3}n,\ldots,\frac{2}{5}n$. We ensure that the size of this subset is $\Omega(N)$ for $N = \binom{n}{k}$. This proves the lower bound.

First we remark that $|A_{i_1,\ldots,i_k} \cap A_i| \leq k$, if $i \notin \{i_1,\ldots,i_k\}$. This has the following consequences. Since (for large n) $p \geq k+2$, the inputs from $f_{n,k}^{-1}(1)$ have a Hamming distance of at least 2 and each 1-path contains n inner nodes. Moreover, an input a' which is the characteristic vector of A' such that $|A' \cap A_i| \geq k+1$ and $A_i \not\subseteq A'$ has the property that $f_{n,k}(a') = 0$.

As next step, we prove that many 1-paths split early. Let I contain the indices of the first $\frac{1}{3}n$ variables according to π. The average size of all $B_i \cap I$ equals $\frac{1}{3}p$ and (for large n) there are two different blocks B_i and B_j such that $|B_i \cap I| \geq \frac{1}{4}p$ and $|B_j \cap I| \geq \frac{1}{4}p$. There are at least $\binom{p/4}{k}\binom{p/4}{k} = \Omega(N)$ (remember that k is a constant) possibilities to choose k elements $i_1,\ldots,i_k$ from $B_i \cap I$ and k elements $j_1,\ldots,j_k$ from $B_j \cap I$. We identify each such choice with a unique minterm. The pair (i_r, j_r) determines by the properties of the A-sets uniquely a set A_{m_r} such that $i_r, j_r \in A_{m_r}$. Since $A_{m_1,\ldots,m_k} \cap B_i = \{i_1,\ldots,i_k\}$ and $A_{m_1,\ldots,m_k} \cap B_j = \{j_1,\ldots,j_k\}$, different choices lead to different 1-paths. Let P' be the set of the chosen $\Omega(N)$ 1-paths. Let us consider two different of these 1-paths or minterms. They correspond to the choices $i_1,\ldots,i_k,j_1,\ldots,j_k$ and $i'_1,\ldots,i'_k,j'_1,\ldots,j'_k$ and w.l.o.g. $i_1 \notin \{i'_1,\ldots,i'_k\}$. The variable x_{i_1} is tested on one of the first $\frac{1}{3}n$ levels and the first minterm chooses a 1-edge on this level and the second one a 0-edge. Hence, the paths from P' split before or at level $\frac{1}{3}n$.

As final step, we prove that many 1-paths from P' do not merge again before level $\frac{2}{5}n$. Let I^* contain the indices of the first $\frac{2}{5}n$ variables according to π. Let r be the number of rich sets A_i, i.e., sets where $|A_i \cap I^*| \geq p-k$. We prove by contradiction that $r \leq p-k-1$. We assume that $|A_{i_1} \cap I^*| \geq p-k,\ldots,$ $|A_{i_{p-k}} \cap I^*| \geq p-k$. Since $|A_{i_1} \cap A_{i_2}| \leq 1$, $|(A_{i_1} \cup A_{i_2}) \cap I^*| \geq (p-k)+(p-k-1)$. In the same way, we conclude (for large n) that $|(A_{i_1} \cup \cdots \cup A_{i_{p-k}}) \cap I^*| \geq (p-k)+(p-k-1)+\cdots+1 \geq \frac{1}{2}(p-k)^2 > \frac{2}{5}n$ in contradiction to $|I^*| = \frac{2}{5}n$.

Let $P'' \subseteq P$ be the set of 1-paths corresponding to sets $A_{i_1,\ldots,i_k}$ such that all A_{i_r} are poor, i.e., not rich. The number of rich A-sets has been shown to be

at most $p-k-1$. Hence, we have more than $n-p$ poor sets and more than $\binom{n-p}{k}$ sets $A_{i_1,\dots,i_k}$ consisting of poor sets only. Since $n-p=n-o(n)$ and k is a constant, $|P''|>\binom{n-p}{k}=N-o(N)$. P' and P'' are subsets of P where $|P|=N$, $|P'|=\Omega(N)$, and $|P''|=N-o(N)$. Hence, $|P'\cap P''|=\Omega(N)$. We consider two different paths p_1 and p_2 from $P'\cap P''$. They have split before or at level $\frac{1}{3}n$. We assume w.l.o.g. that p_1 and p_2 correspond to $A_{i_1,\dots,i_k}$ and $A_{j_1,\dots,j_k}$ and split on the x_i-level, $i\in I$, where $i\in A_{i_1,\dots,i_k}$ (w.l.o.g. $i\in A_{i_1}$) and $i\notin A_{j_1,\dots,j_k}$. Now we assume that p_1 and p_2 share the node v on one of the levels between $\frac{1}{3}n$ and $\frac{2}{5}n$. Then the path p^* following p_2 from the source to v and p_1 from v also is a 1-path corresponding to an input a' which is the characteristic vector of some set A'. Since A_{i_1} is poor, at least $k+1$ variables $x_r, r\in A_{i_1}$, are tested positively on p^*, namely on that part of p_1 which starts at v. Hence, $|A'\cap A_{i_1}|\geq k+1$. Since x_i is tested negatively on p^*, namely on that part of p_2 which stops at v, $A_{i_1}\not\subseteq A'$ and $f_{n,k}(a')=0$ (as shown above) in contradiction to the construction of p^* as 1-path. This proves Theorem 5. □

References

1. Ajtai, M. (1999). A non-linear time lower bound for Boolean branching programs. 40. FOCS, 60–70.
2. Bollig, B. and Wegener, I. (1996). Improving the variable ordering of OBDDs is NP-complete. IEEE Trans. on Computers 45(9), 993–1002.
3. Bollig, B. and Wegener, I. (1999). Asymptotically optimal bounds for OBDDs and the solution of some basic OBDD problems. Electronic Colloquium on Computational Complexity (ECCC), Report # TR99-048.
4. Bryant, R. E. (1986). Graph-based algorithms for Boolean manipulation. IEEE Trans. on Computers 35, 677–691.
5. Hromkovič, J. (1997). *Communication Complexity and Parallel Computing.* Springer.
6. Kovari, T. Sós, V., and Turán, P. (1954). On a problem of K. Zarankiewicz. Colloquium Mathematicum 3, 50–57.
7. Kushilevitz, E. and Nisan, N. (1997). *Communication Complexity.* Cambridge University Press.
8. Minato, S. (1993). Zero-suppressed BDDs for set manipulation in combinatorial problems. 30. DAC, 272–277.
9. Ochi, H., Yasuoka, K., and Yajima, S. (1993). Breadth-first manipulation of very large binary-decision diagrams. ICCAD '93, 48–55.
10. Rudell, R. (1993). Dynamic variable ordering for ordered binary decision diagrams. ICCAD '93, 42–47.
11. Sieling, D. (1998). On the existence of polynomial time approximation schemes for OBDD minimization. STACS '98, LNCS 1373, 205–215.
12. Sieling, D. and Wegener, I. (1993). NC-algorithms for operations on binary decision diagrams. Parallel Processing Letters 48, 139–144.
13. Somenzi, F. (1998). CUDD: CU decision diagram package release 2.3.0. Techn. Rep. of the University of Colorado at Boulder.
14. Wegener, I. (1987). *The Complexity of Boolean Functions.* Wiley-Teubner.
15. Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams - Theory and Applications.* SIAM Monographs in Discrete Mathematics and Applications.

Measures of Nondeterminism in Finite Automata

Juraj Hromkovič[1⋆], Juhani Karhumäki[2*], Hartmut Klauck[3], Georg Schnitger[3], and Sebastian Seibert[1*]

[1] Lehrstuhl für Informatik I, RWTH Aachen, Ahornstraße 55, 52074 Aachen, Germany
[2] Department of Mathematics, University of Turku, Finland
[3] FB Informatik, Johann-Wolfgang-Goethe-Universität, 60054 Frankfurt am Main, Germany

Abstract. While deterministic finite automata seem to be well understood, surprisingly many important problems concerning nondeterministic finite automata (nfa's) remain open. One such problem area is the study of different measures of nondeterminism in finite automata. Our results are:

1. There is an exponential gap in the number of states between unambiguous nfa's and general nfa's. Moreover, deterministic communication complexity provides lower bounds on the size of unambiguous nfa's.
2. For an nfa A we consider the complexity measures $advice_A(n)$ as the number of advice bits, $ambig_A(n)$ as the number of accepting computations, and $leaf_A(n)$ as the number of computations for worst case inputs of size n. These measures are correlated as follows (assuming that the nfa A has at most one "terminally rejecting" state): $advice_A(n), ambig_A(n) \leq leaf_A(n) \leq O(advice_A(n) \cdot ambig_A(n))$.
3. $leaf_A(n)$ is always either a constant, between linear and polynomial in n, or exponential in n.
4. There is a language for which there is an exponential size gap between nfa's with exponential leaf number/ambiguity and nfa's with polynomial leaf number/ambiguity. There also is a family of languages KON_{m^2} such that there is an exponential size gap between nfa's with polynomial leaf number/ambiguity and nfa's with ambiguity m.

Keywords: finite automata, nondeterminism, limited ambiguity, descriptional complexity, communication complexity.

1 Introduction

In this paper the classical models of one-way finite automata (dfa's) and their nondeterministic counterparts (nfa's) are investigated. While the structure and

⋆ Supported by DFG grant Hr-1413-2 and Finland-Germany Project "Descriptional Complexity and Efficient Transformations of Formal Languages over Words, Trees, and Graphs".

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 199–210, 2000.
© Springer-Verlag Berlin Heidelberg 2000

fundamental properties of dfa's are well understood, this is not the case for nfa's. For instance, we have efficient algorithms for constructing minimal dfa's, but the complexity of approximating the size of a minimal nfa is still unresolved (whereas finding a minimal nfa solves a PSPACE complete problem). Hromkovič, Seibert and Wilke [HSW97] proved that the gap between the length of regular expressions and the number of edges of corresponding nfa's is between $n \log^2 n$ and $n \log n$, but the exact gap is unknown. Another principal open question is to determine whether there is an exponential gap between two-way deterministic finite automata and two-way nondeterministic ones. The last partially successful attack on this problem was done in the late seventies by Sipser [S80], who established an exponential gap between determinism and nondeterminism for so-called sweeping automata (the property of sweeping is essential [M80]).
Our main goal is to contribute to a better understanding of the power of nondeterminism in finite automata [MF71]. We focus on the following problems:

1. The best known method for proving lower bounds on the size of minimal nfa's is based on nondeterministic communication complexity [H97]. All other known methods are special cases of this method. Are there methods that provide better lower bounds at least for some languages? How can one prove lower bounds on the size of unambiguous nfa's (unfa's), that is nfa's which have at most one accepting computation for every word?
2. It is a well known fact that there is an exponential gap between the sizes of minimal dfa's and nfa's for some regular languages. This is even known for dfa's and unfa's. But, it is open whether there exists an exponential gap between unfa's and nfa's, i.e., whether unambiguousness is a real restriction on nondeterminism.
3. The degree of nondeterminism is measured in the literature in three different ways. Let A be an nfa. The first measure $advice_A(n)$ equals the number of advice bits for inputs of length n, i.e., the maximum number of nondeterministic guesses in computations for inputs of length n. The second measure $leaf_A(n)$ determines the maximum number of computations for inputs of length n. $ambig_A(n)$ as the third measure equals the maximum number of accepting computations for inputs of length at most n. Obviously the second and third measure may be exponential in the first. The question is whether the measures are correlated.

To attack these problems we establish some new bridges between automata theory and communication complexity. This approach leads to contributions in the study of the tradeoff between the size and the degree of nondeterminism of nfa's. Communication complexity theory contains deep results about the nature of nondeterminism (see e.g., [KNSW94] and [HS96]) using the combinatorial structure of the communication matrix. These results can be applied to finite automata. Our main contributions are as follows:

1. Let $cc(L)$ resp. $ncc(L)$ denote the deterministic resp. nondeterministic communication complexity of L. First we show that there are regular languages L for which there is an exponential gap between $2^{ncc(L)}$ and the minimal size of

nfa's for L. This means, that the lower bound method based on communication complexity may be very weak. Then we show as a somewhat surprising result that $2^{\sqrt{cc(L)}/k} - 2$ is a lower bound on the size of nfa's with ambiguity k for L. We furthermore show that $rank(M)^{1/k} - 1$ is a lower bound for the number of states for nfa's with ambiguity k, where M is a communication matrix associated with L. It is possible that this lower bound is always better than the first, see [KN97] for a discussion of the quality of the so-called rank lower bound on communication complexity.
As a corollary we show that there is a sequence of regular languages NID_m such that the size of a minimal nfa is linear in m, while the size of every unfa for NID_m is exponential in m.

2. We establish (assuming that the nfa A has at most one "terminally rejecting" state) the relation

$$advice_A(n), ambig(n)_A \leq leaf_A(n) \leq O(advice_A(n) \cdot ambig_A(n)).$$

Furthermore we show that $leaf_A(n)$ is always either a constant, between linear and polynomial, or at least exponential in the input length.

3. We show that there is a regular language such that there is an exponential gap between the size of nfa's with exponential ambiguity, and nfa's with polynomial ambiguity. This result is obtained by showing that small nfa's with polynomial ambiguity for the Kleene closure $(L\#)^*$ imply small unfa's that work correctly on a polynomial fraction of inputs.
Furthermore we describe a sequence of languages KON_{m^2} such that there is an exponential size gap between nfa's with polynomial ambiguity and nfa's with ambiguity m. KON_m is a candidate for proving a size gap between nfa's with polynomial ambiguity and nfa's with arbitrary constant ambiguity, even when this ambiguity is larger than the optimal nfa size m.

This paper is organized as follows. In section 2 we give the basic definitions and fix the notation. Section 3 is devoted to the investigation of the relation between the size of nfa's and communication complexity, and it includes the results of 1). Section 4 is devoted to the study of the relation between different measures of nondeterminism in finite automata, and presents the remaining results.

2 Definitions and Preliminaries

We consider the standard models of (one-way) finite automata (dfa's) and (one-way) nondeterministic finite automata (nfa's). For every automaton A, $L(A)$ denotes the language accepted by A. The number of states of A is called the size of A and denoted $size_A$. For every regular language L we denote the size of the minimal dfa for L by $s(L)$ and the size of minimal nfa's accepting L by $ns(L)$. For any nfa A and any input x we use the computation tree $T_{A,x}$ to represent all computations of A on x. Obviously the number of leaves of $T_{A,x}$ is the number of different computations of A on x.

The ambiguity of an nfa A on input x is the number of accepting computations of A on x, i.e., the number of accepting leaves of $T_{A,x}$. If the nfa A has ambiguity one for all inputs, then A is called an unambiguous nfa (unfa) and $uns(L)$ denotes the size of a minimal unfa accepting L. More generally, if an nfa A has ambiguity at most k for all inputs then A is called a k-ambiguous nfa and $ns_k(L)$ denotes the size of a minimal k-ambiguous nfa accepting L.
For every nfa A we measure the degree of nondeterminism in the following ways. Let Σ denote the alphabet of A. For every input $x \in \Sigma^*$ and for every computation C of A on x we define $advice(C)$ as the number of nondeterministic choices during the computation C, i.e., the number of nodes on the path of C in $T_{A,x}$ which have more than one successor. Then

$$advice_A(x) = \max\{advice(C) | C \text{ is a computation of } A \text{ on } x\}$$

$$\text{and } advice_A(n) = \max\{advice(x) | x \in \Sigma^n\}.$$

For every $x \in \Sigma^*$ we define $leaf_A(x)$ as the number of leaves of $T_{A,x}$ and set

$$leaf_A(n) = \max\{leaf(x) | x \in \Sigma^n\}.$$

For every $x \in \Sigma^*$ we define $ambig_A(x)$ as the number of accepting leaves of $T_{A,x}$ and set

$$ambig_A(n) = \max\{ambig(x) | x \in \Sigma^{\leq n}\}.$$

Since a language need not contain words of all lengths we define ambiguity over all words of length at most n which makes the measure monotone. Observe that the leaf and advice measures are monotone as well.
Note that different definitions have been used by other authors, see e.g. [GKW90], [GLW92], where the number of advice bits is maximized over all inputs and minimized over all accepting computations on those inputs. In this case there are nfa's which use more than constant but less than linear (in the input length) advice bits, but this behavior is not known to be possible for minimal nfa's.
To prove lower bounds on the size of finite automata we shall use two-party communication complexity. This widely studied measure was introduced by Yao [Y79] and is the subject of two monographs [H97], [KN97]. Let $L \subseteq X \times Y$ be a (possibly infinite) language. A two-party communication protocol works on two computers C_I and C_{II} with unbounded computational power and a communication link between them. At the beginning of the computation C_I receives an input $x \in X$, and C_{II} an input $y \in Y$. Then the computers exchange binary messages according to a fixed protocol until one of them knows whether $xy \in L$. $cc(L)$ resp. $ncc(L)$ is the minimum over all deterministic resp. nondeterminis tic protocols of the worst case number of bits exchanged for any input. Note that for $X, Y = \Sigma^*$ this definition yields the uniform version of communication complexity [DHRS97], for $X, Y = \{0,1\}^n$ the standard version. Unless stated otherwise we use $X, Y = \Sigma^*$.

A one-way protocol P is a protocol, in which only one message is sent from C_I to C_{II}, who is then able to compute the answer. The one-way communication complexity of L is the communication complexity of the best one-way protocol for L. The message complexity $mc(P)$ for a deterministic one-way protocol P counts the number of different messages sent by P. The message complexity of L is the message complexity of the best one-way protocol for L. The nondeterministic message complexity of L, $nmc(L)$, is the minimum over all nondeterministic protocols recognizing L of the number of messages exchanged over all inputs. It is well known [PS84] that for nondeterministic communication one-way protocols are optimal.
Bounded ambiguity protocols have been investigated in [Y91], [L90], [KNSW94], protocols with bounded advice in [HS96],[K98].

3 Communication Complexity and Finite Automata

Ďuriš, Hromkovič, Rolim, and Schnitger [DHRS97] (see also [H86]) observed that the minimal number of states of a dfa recognizing L can be characterized by the communication complexity of L.

Fact 1 *For every regular language L: $mc(L) = s(L)$.*

The lower bound $s(L) \geq mc(L)$ can easily be generalized to nondeterministic and probabilistic automata.
Nondeterministic communication complexity seems to provide the best lower bounds on $ns(L)$. All previously known techniques like the fooling set approach are special cases of this approach. Moreover the fooling set method, which covers all previous efforts in proving lower bounds on $ns(L)$, can (for some languages) provide exponentially smaller lower bounds than the method based on nondeterministic communication complexity [DHS96].
The first question is therefore whether $nmc(L)$ can be used to approximate $ns(L)$. Unfortunately this is not possible.

Lemma 1 *There exists a regular language $PART$ such that*

$$ns(PART) \geq 2^{\Omega(\sqrt{nmc(PART)})}.$$

Proof Sketch: Let $PART = \{xyz : |x| = |y| = |z| = n, \text{ and } x \neq z \vee x = y\}$. First we describe a nondeterministic protocol for $PART$ using $O(n^2)$ messages. Players C_I and C_{II} compute the lengths l_I, l_{II} of their input. C_I communicates l_I and C_{II} rejects when $l_I + l_{II} \neq 3n$. So we assume that $l_I + l_{II} = 3n$ in the following.
Case 1: $l_I \leq n$.
C_I chooses a position $1 \leq i \leq l_I$ and communicates i, x_i, l_I. C_{II} accepts, if $x_i \neq z_i$. Otherwise C_{II} accepts if and only if $y = z$.
Note that if $x \neq z$, then there is an accepting computation. If however $x = z$, then C_{II} accepts iff $y = z$, that is iff $x = y$.

The other cases $n < l_I \leq 2n$ and $2n < l_I \leq 3n$ are handled similarly using $O(n^2)$ messages.
We give the following lower bound on $ns(PART)$. Assume that the input is xyx. Then the nfa A has to test whether $x = y$. A can be simulated by a 2 round nondeterministic protocol, where C_I holds x and C_{II} holds y, using communication $2 \log size_A$. Then $size_A$ must be at least $2^{n/2}$. □
To find lower bound methods for $ns(L)$ that provide results at most polynomially smaller than $ns(L)$ is one of the central open problems on finite automata. In the following we concentrate on lower bounds for nfa's with constant ambiguity. Even for unambiguous automata no nontrivial method for proving lower bounds has been known up to now.
To present our method we represent regular languages as communication matrices of infinite size. For an alphabet Σ and sets $X, Y \subseteq \Sigma^*$ the communication matrix of a language L is the infinite Boolean matrix $M(L, \Sigma, X, Y) = \{a_{x,y}\}$ for $x \in X, y \in Y$, where $a_{x,y} = 1$ iff $xy \in L$. We will use the shorter notation $M(L)$, where Σ, X, Y are clear from the context.
In [DHRS97] it was proved that the number of different rows of $M(L)$ equals $mc(L)$ which in turn equals $s(L)$. Since the number of different rows is finite for all regular languages we can define $rank_{\mathbb{Q}}(M(L))$ as the rank of $M(L)$ over the rational numbers $\mathbb{Q}$.

Theorem 1 *For every regular $L \subseteq \Sigma^*$*
a) $uns(L) \geq rank_{\mathbb{Q}}(M(L))$
b) $ns_k(L) \geq rank_{\mathbb{Q}}(M(L))^{1/k} - 1$.
c) $ns_k(L) \geq 2^{\sqrt{cc(L))}/k} - 2$.

Proof: Let A be an optimal unfa for L. A can be simulated by a one-way nondeterministic protocol as follows: C_I simulates A on its input and communicates the obtained state. C_{II} continues the simulation and accepts/rejects accordingly. Obviously the number of messages is equal to $size_A$ and the protocol works with unambiguous nondeterminism.
It is easy to see that the messages of the protocol correspond to $size_A$ submatrices of the matrix $M(L)$ covering all ones exactly once. Hence the rank is at most $size_A$ and we have shown a), which is related to the rank lower bound on communication complexity [MS82].
For b) observe that the above simulation induces a cover of the ones in $M(L)$ so that each one is covered at most k times. By the following fact from [KNSW94] we are done:

Fact 2 *Let $\kappa_r(M)$ denote the minimal size of a set of submatrices covering the ones of a Boolean matrix M so that each is covered at most r times. Then*

$$(1 + \kappa_r(M))^r \geq rank(M).$$

For the other claim again simulate A by a one-way k-ambiguous nondeterministic protocol with $size_A$ messages.

Results of [KNSW94] (see also [L90], [Y91]) imply that a k-ambiguous nondeterministic one-way protocol with m messages can be simulated by a deterministic two-way protocol with communication $\log(m^k+1)\cdot k\cdot\log(m+2)$. Thus $cc(L)\leq\log(size_A^k+1)\cdot k\cdot\log(size_A+2)\leq\log^2((size_A+2)^k)$ and c) follows. □
Before giving an application of the lower bound method we point out that neither $2^{\sqrt{cc(L)}}$ nor $rank_{\mathbb{Q}}(M(L))$ is a lower bound method capable of proving polynomially tight lower bounds on the minimal size of unfa's for all languages. In the first case this is trivial, in the second case it follows from a modification of a result separating rank from communication complexity (see [KN97]). But the lower bounds may still be pseudopolynomial.
Now we apply theorem 1 in order to prove an exponential gap between $ns(L)$ and $uns(L)$ for a regular language.

Corollary 1 *There is a language NID_m with the following properties:*
a) *NID_m can be recognized by an nfa A with ambiguity $O(m)$ and size $O(m)$*
b) *Any nfa with ambiguity k for NID_m has size at least $2^{m/k}-1$, and in particular any unfa for NID_m must have 2^m states.*
c) *No nfa with ambiguity $o(m/\log m)$ for NID_m has polynomial size in m.*

Proof: Let $NID_m=\{u\in\{0,1\}^*|\exists i: u_i\neq u_{i+m}\}$.
a) First the nfa guesses a residue i modulo m, and then checks whether there is a position $p\equiv i \bmod m$ with $u_p\neq u_{p+m}$.
b) Observe that the submatrix spanned by all words u and v with $u,v\in\{0,1\}^m$ is the "complement" of the $2^m\times 2^m$ identity matrix. The result now follows from parts a) and b) of theorem 1.
c) is an immediate consequence of b). □

4 Degrees of Nondeterminism in Finite Automata

It is easy to see that $advice_A(n)\leq leaf_A(n)\leq 2^{O(advice_A(n))}$ and also that $ambig_A(n)\leq leaf_A(n)$ for every nfa A. The aim of this section is to investigate whether stronger relations between these measures hold.

Lemma 2 *For all nfa A either*
a) *$advice_A(n)\leq size_A$ and $leaf_A(n)\leq size_A^{size_A}$ or*
b) *$advice_A(n)\geq n/size_A-1$ and $leaf_A(n)\geq n/size_A-1$.*

Proof: If some reachable state q of A belongs to a cycle in A and if q has two edges with the same label originating from it such that one of these edges belongs to the cycle, then $advice_A(n)\geq(n-size_A)/size_A\geq n/size_A-1$. Otherwise for all words all states with a nondeterministic decision are traversed at most once. □
Our next lemma relates the $leaf$ function to ambiguity. A state q of an nfa A is called *terminally rejecting*, if there is no word and no computation of A, such that A accepts when starting in q, i.e., $\delta^*(q,v)$ contains no accepting state for any word v. Clearly there is at most one terminally rejecting state in a minimal automaton, because otherwise these states can be joined reducing the size. Call all other states of A *undecided*.

Lemma 3 *Every nfa A with at most one terminally rejecting state satisfies*

$$leaf_A(x) \leq ambig_A(|x| + size_A) \cdot |x| \cdot size_A + 1$$

for all x.

Proof: Let $k = ambig_A(|x| + size_A)$. If the computation tree consists only of nodes marked with the terminally rejecting state, then the tree consists of just one leaf and the claim is trivial. For the general case, consider a level of the computation tree of A on x that is not the root level. Assume that the level contains more that $k \cdot size_A$ nodes labeled with undecided states (called undecided nodes). Then one undecided state q must appear at least $k+1$ times on this level. There are $k+1$ computations of A on a prefix of x such that q is reached. If q is accepting, then the prefix of x is accepted with $k+1$ computations, a contradiction, since $ambig_A$ is monotone. If q is rejecting, but undecided, then there is a word v of length at most $size_A$ such that v is accepted by some computation of A starting in q. But then the prefix of x concatenated with v is accepted by at least $k+1$ computations, a contradiction.
Thus each level of the tree that is not the root level contains at most $k \cdot size_A$ undecided nodes. Overall there are at most $|x| \cdot k \cdot size_A + 1$ undecided nodes.
Observe that each node has at most one terminally rejecting child. Thus the number of terminally rejecting leaves is equal to the number of undecided nodes that have a terminally rejecting child. Hence the number of terminally rejecting leaves is at most the number of undecided nodes minus the number of undecided leaves. Thus the overall number of leaves is at most the number of terminally rejecting leaves plus the number of undecided leaves which is at most the number of undecided nodes. So overall there are at most $k \cdot |x| \cdot size_A + 1$ leaves. □

Theorem 2 *Every nfa A with at most one terminally rejecting state satisfies*

$$advice_A(n), ambig_A(n) \leq leaf_A(n) \leq O(ambig_A(n) \cdot advice_A(n)).$$

Especially for any such unfa: $advice_A(n) = \Theta(leaf_A(n))$.

Proof: Observe that for all n: $ambig_A(n) = \Omega(ambig_A(n+O(1)))$, since $ambig_A$ is monotone and at most exponential. □
Next we further investigate the growth of the leaf function.

Lemma 4 *For every nfa A either $leaf_A(n) \leq (n \cdot size_A)^{size_A}$ or $leaf_A(n) \geq 2^{\Omega(n)}$.*

Proof: Assume that an nfa A contains some state q, such that q can be entered on two different paths starting in q, where each path is labeled with the same word w. It is not hard to show that in this case there are two different paths from q to q labeled with a word w of length $size_A^2 - 1$. Then the computation tree of uw^m (where u leads from the starting state to q) has at least $2^m \geq 2^{(n-size_A)/size_A^2}$ leafs, where $n = |uw^m|$.

Now assume that A does not contain such a state. Then for each nondeterministic state q (i.e., a state with more than one successor for the same letter) and any computation tree the following holds: If q is the label of a vertex v, then q appears in each level of the subtree of v at most once.
We prove by induction over the number k ($k \leq size_A$) of different nondeterministic states in a computation tree that the number of leafs is at most $(n \cdot size_A)^k$.
The claim is certainly true if there are no nondeterministic states.
Assume that there are k nondeterministic states, with some state q_1 appearing first in the tree. Observe that no level in the entire computation tree contains q_1 more than once.
For each occurence of q_1 in the computation tree fix some child, so that the overall number of leaves is maximized. We get a tree with one nondeterministic state less, and by the induction hypothesis this tree has at most $(n \cdot size_A)^{k-1}$ leaves.
Since q_1 appears at most once on each level and since there are at most $size_A$ children of q_1 on each level, there are at most $(n \cdot size_A)^k$ leaves. □
Lemma 2 and 4 give us

Theorem 3 *For all A: $leaf_A(n)$ is bounded by a constant, or is between linear and polynomial in n, or is $2^{\Theta(n)}$.*

Now we consider the difference between polynomial and exponential ambiguity resp. polynomial and exponential leaf number. We show that languages which have small automata of polynomial ambiguity are related to the concatenation of languages having small unfa's. If the language is a Kleene closure, then one unfa accepts a large subset. Compare [GKW90], where Kleene closures are shown to be recognizable as efficient by nfa's with constant advice as by dfa's.

Theorem 4 a) *Let L be an infinite regular language and A some nfa for L with polynomial ambiguity. Then there are $k \leq size_A$ languages L_i such that $L_1 \cdots L_k \subseteq L$, L_i is recognizable by an unfa with $O(size_A)$ states, and*

$$\frac{|L_1 \cdots L_k \cap \Sigma^n|}{|L \cap \Sigma^n|} = \Omega(1) \text{ for infinitely many } n.$$

b) *Let $L = (K\#)^*$ for a regular language K not using the letter $\#$ and let A be some nfa for L with polynomial ambiguity. Then for all m there is an unfa A' with $O(size_A)$ states that decides $L' \subseteq L$ such that for infinitely many n*

$$\frac{|L' \cap (\Sigma^m \cap K)\#)^n|}{|((\Sigma^m \cap K)\#)^n|} = \Omega(1/poly(n)).$$

Proof Sketch: **a)** Define the ambiguity graph of A in the following way: the nodes are the (reachable) states of A and there is an edge from q_i to q_j if there are two paths from q_i to q_j in A with the same label sequence. Note that the ambiguity graph is acyclic iff the ambiguity of A is polynomially bounded as we have seen in the proof of lemma 4.

We construct now a unfa $A_{i,j,k}$ which accepts those words that lead in A from q_i to q_j and then via one edge to q_k. Here we assume that the longest path from q_i to q_k in the ambiguity graph consists of one edge and q_j is reachable from q_i in A, but not in the ambiguity graph. Moreover, we demand that there is an edge in A from q_j to q_k.
The states of $A_{i,j,k}$ are the states reachable in A from q_i, but not reachable in the ambiguity graph from q_i, plus the state q_k. The edges are as in A except that the only edges to q_k come from q_j. q_i is the start. Accepting state is q_k. $L_{i,j,k}$ is the language accepted by $A_{i,j,k}$.
Now consider the words $w \in L \cap \Sigma^n$. Each such word is accepted on some path in A leading from q_0 to some accepting state q_a. Fix one such accepting state so that a constant fraction of all words w is accepted and make the other accepting states rejecting. On an accepting path for w the states appear without violating the topological ordering of the ambiguity graph. So we may fix a sequence of states $q_0, q_{i_1}, \ldots, q_a$ so that $w \in L_{0,i_1,i_2} L_{i_2,i_3,i_4} \cdots L_{i_{2k-2},i_{2k-1},a}$. Since there are only finitely many such sequences we are done.
b) Similar to a) we get k languages $L_1, \ldots, L_k$ decidable by small unfa's A_i, such that $\frac{|L_1 \cdots L_k \cap ((\Sigma^m \cap K)\#)^n|}{|((\Sigma^m \cap K)\#)^n|} = \Omega(1)$ for infinitely many n.
A partition of the letters of words in $(\Sigma^m\#)^n$ is given by mapping the nm letters to the k unfa's. There are at most $\binom{n}{k-1} \cdot (m+1)^{k-1}$ possible partitions. So some partition must be consistent with accepting paths for a fraction of $1/poly(n)$ of $((\Sigma^m \cap K)\#)^n$. Fix one such partition. Then for each words $w \in (\Sigma^m\#)^n$ an unfa is responsible for some prefix u, followed by a concatenation of words of the form $\#\Sigma^m$, and finally a word of the form $\#v$. For all i we fix a prefix u_i, a suffix v_i, and states q_i, q_i' entered when reading the first and final occurence of $\#$, such that as many words from $((\Sigma^m \cap K)\#)^n$ as possible are accepted under this fixing. At least a fraction of $size^{-k}/2^{O(mk)} = 1/poly(n)$ of $((\Sigma^m \cap K)\#)^n$ has accepting paths consistent with this fixing.
If any A_i accepts less than a polynomial fraction (compared to the projection of $(\Sigma^m \cap K)\#)^n$ to the responsibility region of A_i) then overall less than a polynomial fraction is accepted. Hence one A_i can be found, where from q_i a polynomial fraction of words in $(\Sigma^m \cap K)\#)^{n/k}$ leads to non-terminally rejecting states in A_i. Making one non-terminally rejecting state reached by a $\#$ edge accepting and removing the original accepting states yields an unfa that accepts the desired subset for infinitely many n. □

Corollary 2 *There is a family of languages KL_m such that KL_m can be recognized by an nfa with advice $\Theta(n)$, leaf $2^{\Theta(n)}$ and size $poly(m)$, while every nfa with polynomial leaf number/ambiguity needs size $2^{\Omega(m)}$ to recognize KL_m.*

Proof: Let $LNDISJ_m = \{x_1 \cdots x_m \cdot y_1 \cdots y_m | x_i, y_i$ encode elements from a size m^{32} universe and the sets $\cup_i x_i$ and $\cup_i y_i$ intersect nontrivially$\}$. Then let $KL_m = (LNDISJ_m\#)^*$.
Given a polynomial ambiguity nfa for KL_m we get an unfa accepting a fraction of $1/poly(n)$ of $(LNDISJ_m\#)^n$ for infinitely many n by theorem 4b). Then we simulate the unfa by a nondeterministic communication protocol, where player C_I receives all x and player C_{II} all y inputs. The protocol needs $O(n \cdot \log size_A)$

bits to decide correctly on a $1/poly(n)$ fraction of $(LNDISJ_m\#)^n$ and has unambiguous nondeterminism. A result from [HS96] implies that this task needs communica tion $\Omega(nm)$ and thus $size_A \geq 2^{\Omega(m)}$. □

So we have a strong separation between the size of automata with polynomial ambiguity and the size of automata with exponential ambiguity. The situation seems to be more complicated, if one compares constant and polynomial ambiguity. Here we can only show that there is a family KON_{m^2} of languages with small size nfa's of polynomial ambiguity, while nfa's of ambiguity m are exponentially larger. In the following theorem we describe a candidate for a language that has efficient nfa's only when ambigui ty is polynomial. Furthermore the language exhibits an almost optimal gap between the size of unfa's and polynomial ambiguity nfa's. In the (omitted) proof the rank of the communication matrix of KON_m is shown to be large by a reduction from the disjointness problem.

Theorem 5 *Let $KON_m = \{0,1\}^*0M_m0\{0,1\}^*$, where M_m contains all words in $\{0,1\}^*$ with a number of ones that is divisible by m. KON_m can be recognized by an nfa A with $ambig_A(n), leaf_A(n) = \Theta(n)$ and size $m+2$, while any nfa with ambiguity k for KON_m needs at least $2^{(m-1)/k} - 2$ states.*

For every m the language KON_{m^2} of theorem 5 can be recognized with size $O(m^2)$, leaf number and ambiguity $\Theta(n)$, and advice $\Theta(n)$, while every $m-$ambiguous nfa has size $2^{\Omega(m)}$. We conjecture that the language KON_m cannot be decided by nfa's with constant ambiguity (even larger than m) and size $poly(m)$.

5 Conclusions and Open Problems

We have shown that communication complexity can be used to prove size lower bounds for nfa's with small ambiguity. This approach is limited, because for nontrivial bounds ambiguity has to be smaller than the size of a minimal nfa. Is it possible to prove lower bounds for automata with arbitrarily large constant ambiguity, when small equivalent polynomial ambiguity automata exist?

In this context, it would be also of interest to investigate the fine structure of languages with regard to constant ambiguity. At best one could show exponential differences between the number of states for ambiguity k and the number of states for ambiguity $k+1$. Observe however, that such an increase in power is impossible provided that the size of unfa's does not increase substantially under complementation [K00]. Analogous questions apply to polynomial and exponential ambiguity.

Are there automata with nonconstant but sublinear ambiguity? A negative answer establishes theorem 3 also for ambiguity as complexity measure.

Other questions concern the quality of communication as a lower bound method. How far can *rank* resp. $2^{\sqrt{cc(L)}}$ be from the minimal unfa size? Note that the bounds are not polynomially tight. Are there alternative lower bound methods? Finally, what is the complexity of approximating the minimal number of states of an nfa?

References

DHS96. M. Dietzfelbinger, J. Hromkovič, G. Schnitger. A comparison of two lower bound methods for communication complexity. *Theoretical Computer Science*, vol.168, pp.39–51, 1996.

DHRS97. P.Duriš, J. Hromkovič, J.D.P. Rolim, G. Schnitger. Las Vegas Versus Determinism for One-way Communication Complexity, Finite Automata, and Polynomial-time Computations. *Symp. on Theoretical Aspects of Comp. Science*, LNCS 1200, pp. 117–128, 1997.

GKW90. J.Goldstine, C.M.R.Kintala, D.Wotschke. On Measuring Nondeterminism in Regular Languages. *Information and Comput.*, vol.86, pp. 179–194, 1990.

GLW92. J. Goldstine, H. Leung, D. Wotschke. On the Relation between Ambiguity and Nondeterminism in Finite Automata. *Information and Computation*, vol.100, pp. 261–270, 1992.

H86. J. Hromkovič. Relation between Chomsky Hierarchy and Communication Complexity Hier. *Acta Math. Univ. Com.*, vol.48–49, pp. 311-317, 1986.

H97. J.Hromkovič. Communication Complexity and Parallel Computing. Springer, 1997.

HS96. J. Hromkovič, G. Schnitger. Nondeterministic Communication with a Limited Number of Advice Bits. *Proc. 28th ACM Symposium on Theory of Computation.*, pp. 451–560, 1996.

HSW97. J. Hromkovič, S. Seibert, T. Wilke. Translating regular expressions into small ϵ-free nondeterministic finite automata. *Symp. on Theoretical Aspects of Comp. Science*, LNCS 1200, pp. 55–66, 1997.

KNSW94. M. Karchmer, M. Saks, I. Newman, A. Wigderson. Non-deterministic Communication Complexity with few Witnesses. *Journ. of Computer and System Sciences*, vol.49, pp. 247–257, 1994.

K98. H. Klauck. Lower bounds for computation with limited Nondeterminism. *13th IEEE Conference on Computational Complexity*, pp. 141–153, 1998.

K00. H. Klauck. On automata with constant ambiguity. Manuscript.

KN97. E. Kushilevitz, N. Nisan. Communication Complexity. Cambridge University Press, 1997.

L90. L. Lovasz. Communication Complexity: A survey. in: *Paths, Flows, and VLSI Layout*, Springer 1990.

MS82. K. Mehlhorn, E. Schmidt. Las Vegas is better than determinism in VLSI and Distributed Computing. *Proc. 14th ACM Symp. on Theory of Computing*, pp. 330–337, 1982.

MF71. A.R. Meyer, M.J. Fischer. Economy of description by automata, grammars and formal systems, *Proc. 12th Annual Symp. on Switching and automata theory*, pp. 188-191, 1971.

M80. S. Micali. Two-Way Deterministic Finite Automata are Exponentially More Succinct than Sweeping Automata, *Information Proc. Letters*, vol.12, 1981.

PS84. C. Papadimitriou, M. Sipser. Communication Complexity. *Journ. of Computer and System Sciences*, vol.28, pp. 260–269, 1984.

S80. M. Sipser. Lower Bounds on the Size of Sweeping Automata. *Journ. of Computer and System Sciences*, vol.21(2), pp. 195-202, 1980.

Y91. M. Yannakakis. Expressing Combinatorial Optimization Problems by Linear Programs. *Journ. of Comp. and System Sciences*, vol.43, pp. 223–228, 1991.

Y79. A. Yao. Some Complexity Questions Related to Distributed Computing. *Proc. 11th ACM Symp. on Theory of Computing*, pp. 209–213, 1979.

LTL Is Expressively Complete for Mazurkiewicz Traces

Volker Diekert[1] and Paul Gastin[2]

[1] Inst. für Informatik, Universität Stuttgart, Breitwiesenstr. 20-22, D-70565 Stuttgart
diekert@informatik.uni-stuttgart.de
[2] LIAFA, Université Paris 7, 2, place Jussieu, F-75251 Paris Cedex 05
Paul.Gastin@liafa.jussieu.fr

Abstract. A long standing open problem in the theory of (Mazurkiewicz) traces has been the question whether LTL (Linear Time Logic) is expressively complete with respect to the first order theory. We solve this problem positively for finite and infinite traces and for the simplest temporal logic, which is based only on next and until modalities. Similar results were established previously, but they were all weaker, since they used additional past or future modalities. Another feature of our work is that our proof is direct and does not use any reduction to the word case.
Keywords Temporal logics, Mazurkiewicz traces, concurrency

1 Introduction

Nowadays, it is widely accepted that we need to develop methods to verify critical systems. For this, we need formal specifications for the expected behaviors of systems. Most often, these formal specifications are given by temporal logics formulae. The task is even harder when considering concurrent systems. Indeed, the usual approach is to reduce concurrent systems to sequential ones by considering linearizations of concurrent executions. This allows to use techniques and tools developed for sequential systems but introduces a combinatorial explosion which has then to be fought by all means. Instead, one could try to work directly on concurrent systems and this explains why a lot of research has been devoted recently to the study of temporal logics for concurrency. A major aim is to find a temporal logic which is expressive enough to ensure that all desired specification can be formalized.

For sequential systems, Kamp's Theorem [8] states that the linear time logic has the same expressive power as the first order theory of words. Originally Kamp used future and past modalities, but it was established later by Gabbay's separation theorem [6] that past modalities can be avoided.

Trace theory, initiated by Mazurkiewicz is one of the most popular settings to study concurrency. See [3] for the general background of trace theory, and in particular [13] for traces and logic and [7] for infinite traces. It is no surprise that various temporal logics for traces have been extensively studied [1,10,11,12,14,15,16]. A long standing open problem [4,17] was to know

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 211–223, 2000.
© Springer-Verlag Berlin Heidelberg 2000

whether LTL (the basic linear temporal logic) is expressively complete with respect to the first order theory of finite and infinite traces.

The first completeness result for traces is from Ebinger [4]. He proved that a linear temporal logic with both past and future modalities is expressively complete for finite traces but his approach did not cope with infinite traces. Then, Thiagarajan and Walukiewicz [17] proved the completeness both for finite and infinite traces of LTrL, a linear temporal logic with future modalities and past modalities in the restricted form of past constants. These two results were obtained using a reduction to the word case. In [9] it was claimed that LTL is expressively complete for finite traces, but the proof contained a serious mistake. The expressive completeness for a pure future temporal logic (LTL_f) was first established in [2]. The result holds for finite and infinite traces, but LTL_f contains new filter modalities in addition to the usual next and until modalities. Thus, the problem of the expressive completeness of the basic linear temporal logic was still open.

In this paper, we solve this problem positively. Previous expressive completeness results for words and for traces are now formal corollaries of our main theorem. It should be noted that, contrary to most previous works, our proof does not use any reduction to the word case. Instead, we extend the new proof introduced by Wilke for finite words [18] which is based on the well-known fact that first order languages are aperiodic. We follow here the same approach as in [2]. In the former paper filter modalities were used to express some special products of trace languages. Our new proof is a substantial revision of the previous one. We are now able to express the products mentioned above directly with basic formulae without using the filter modalities.

2 Preliminaries

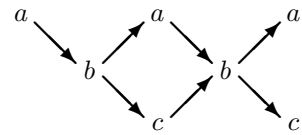
By (Σ, I) we mean a finite *independence alphabet* where Σ denotes a finite alphabet and $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric relation called the *independence relation*. The complementary relation $D = (\Sigma \times \Sigma) \setminus I$ is called the *dependence relation*. The monoid of *finite traces* $\mathbb{M}(\Sigma, I)$ is defined as a quotient monoid with respect to the congruence relation induced by I, i.e., $\mathbb{M}(\Sigma, I) = \Sigma^*/\{\, ab = ba \mid (a,b) \in I \,\}$. We also write $\mathbb{M}$ instead of $\mathbb{M}(\Sigma, I)$. For $A \subseteq \Sigma$ we denote by $\mathbb{M}_A$ the submonoid of $\mathbb{M}(\Sigma, I)$ generated by A:

$$\mathbb{M}_A = \mathbb{M}(A, I \cap A \times A) = \{x \in \mathbb{M}(\Sigma, I) \mid \mathrm{alph}(x) \subseteq A\}.$$

A trace $x \in \mathbb{M}$ is given by a congruence class of a word $a_1 \cdots a_n \in \Sigma^*$ where $a_i \in \Sigma$, $1 \leq n$. By abuse of language, but for simplicity we denote a trace x by one of its representing words $a_1 \cdots a_n$. The number n is called the length of x, it is denoted by $|x|$. For $n = 0$ we obtain the *empty* trace, it is denoted by 1. The *alphabet* $\mathrm{alph}(x)$ of a trace x is the set of letters occurring in x. A *trace language* is a subset $L \subseteq \mathbb{M}$. The concatenation is defined as usual:

$$KL = \{xy \in \mathbb{M} \mid x \in K, y \in L\}.$$

Every trace $a_1 \cdots a_n \in \mathbb{M}$ can be identified with its dependence graph. This is (an isomorphism class of) a node–labeled, acyclic, directed graph $[V, E, \lambda]$, where $V = \{1, \ldots, n\}$ is a set of vertices, each $i \in V$ is labeled by $\lambda(i) = a_i$, and there is an edge $(i, j) \in E$ if and only if both $i < j$ and $(\lambda(i), \lambda(j)) \in D$. In pictures it is common to draw the Hasse diagram only. Thus, all redundant edges are omitted. For instance, let $(\Sigma, D) = a - b - c$, i.e., $I = \{(a, c), (c, a)\}$. Then the trace $x = abcabca$ is given by



An *infinite trace* is an infinite dependence graph $[V, E, \lambda]$ such that for all $j \in V$ the set $\downarrow j = \{i \in V \mid i \leq j\}$ is finite. A *real trace* is a finite or infinite trace. The set of real traces is denoted by $\mathbb{R}(\Sigma, I)$ or simply by $\mathbb{R}$. For a real trace $x = [V, E, \lambda]$ the *alphabet* is $\mathrm{alph}(x) = \lambda^{-1}(V)$ and the *alphabet at infinity* is defined by the set of letters occurring infinitely many times in x, i.e., $\mathrm{alphinf}(x) = \{a \in \Sigma \mid |\lambda^{-1}(a)| = \infty\}$. See [7] for details about infinite traces. Usually we consider real trace languages. If we speak about *finitary* languages, then we refer to subsets of $\mathbb{M}$. As for finite traces, if $A \subseteq \Sigma$ is a subalphabet then we let $\mathbb{R}_A = \{x \in \mathbb{R} \mid \mathrm{alph}(x) \subseteq A\}$.

By $\min(x)$ and $\max(x)$ we refer to the *minimal* and *maximal letters* in the dependence graph. (We shall use the notation $\max(x)$ only for finite traces $x \in \mathbb{M}$.) In the example above $\min(x) = \{a\}$ and $\max(x) = \{a, c\}$. Formally:

$$\begin{aligned} \min(x) &= \{a \in \Sigma \mid x \in a\mathbb{R}\}, \\ \max(x) &= \{a \in \Sigma \mid x \in \mathbb{M}a\}. \end{aligned}$$

For $B \subseteq \Sigma$ and $\# \in \{\subseteq, =, \supseteq, \neq\}$ we define:

$$\begin{aligned} I(B) &= \{a \in \Sigma \mid \forall b \in B : (a, b) \in I\}, \\ D(B) &= \{a \in \Sigma \mid \exists b \in B : (a, b) \in D\}, \\ (\mathrm{Min}\ \#\ B) &= \{x \in \mathbb{R} \mid \min(x)\ \#\ B\}, \\ (\mathrm{Max}\ \#\ B) &= \{x \in \mathbb{M} \mid \max(x)\ \#\ B\}. \end{aligned}$$

Note that $(\mathrm{Max}\ \#\ B)$ denotes a finitary language, whereas $(\mathrm{Min}\ \#\ B)$ is a real trace language. If B happens to be a singleton, then we usually omit braces, e.g. we write $I(b)$ or $(\mathrm{Max} = b)$.

3 Temporal Logic for Traces

The syntax of the temporal logic $\mathrm{LTL}(\Sigma)$ is defined as follows. There are a constant symbol $\bot$ representing *false*, the logical connectives $\neg$ (not) and $\vee$ (or), for each $a \in \Sigma$ a unary operator $\langle a \rangle$, called *next-a*, and a binary operator $\mathbf{U}$, called *until*. Formally, the syntax is given by:

$$\varphi ::= \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle a\rangle\varphi \mid \varphi \,\mathbf{U}\, \varphi,$$

where $a \in \Sigma$.

The semantics is usually defined by saying when some formula φ is satisfied by some real trace z at some configuration (i.e., finite prefix) x; hence by defining $(z, x) \models \varphi$. Since our temporal logic uses future modalities only, we have $(z, x) \models \varphi$ if and only if $(y, 1) \models \varphi$, where y is the unique trace satisfying $z = xy$. Therefore, we do not need to deal with configurations and it is enough to say when a trace satisfies a formula at the empty configuration, denoted simply by $z \models \varphi$. This is done inductively on the formula as follows:

$$\begin{array}{ll} z \not\models \bot, & \\ z \models \neg\varphi & \text{if } z \not\models \varphi, \\ z \models \varphi \vee \psi & \text{if } z \models \varphi \text{ or } z \models \psi, \\ z \models \langle a\rangle\varphi & \text{if } z = ay \text{ and } y \models \varphi, \\ z \models \varphi \,\mathbf{U}\, \psi & \text{if } z = xy,\ x \in \mathbb{M},\ y \models \psi, \text{ and } x = x'x'',\ x'' \neq 1 \text{ implies } x''y \models \varphi. \end{array}$$

As usual, we define $\mathrm{L}_{\mathbb{R}}(\varphi) = \{x \in \mathbb{R} \mid x \models \varphi\}$. We say that a trace language $L \subseteq \mathbb{R}$ is *expressible in* $\mathrm{LTL}(\Sigma)$, if there exists a formula $\varphi \in \mathrm{LTL}(\Sigma)$ such that $L = \mathrm{L}_{\mathbb{R}}(\varphi)$.

Equivalently, we can define inductively the language $\mathrm{L}_{\mathbb{R}}(\varphi)$ as follows:

$$\begin{aligned} \mathrm{L}_{\mathbb{R}}(\bot) &= \emptyset \\ \mathrm{L}_{\mathbb{R}}(\neg\varphi) &= \mathbb{R} \setminus \mathrm{L}_{\mathbb{R}}(\varphi) \\ \mathrm{L}_{\mathbb{R}}(\varphi \vee \psi) &= \mathrm{L}_{\mathbb{R}}(\varphi) \cup \mathrm{L}_{\mathbb{R}}(\psi) \\ \mathrm{L}_{\mathbb{R}}(\langle a\rangle\varphi) &= a\mathrm{L}_{\mathbb{R}}(\varphi) \\ \mathrm{L}_{\mathbb{R}}(\varphi \,\mathbf{U}\, \psi) &= \mathrm{L}_{\mathbb{R}}(\varphi) \,\mathbf{U}\, \mathrm{L}_{\mathbb{R}}(\psi) \end{aligned}$$

where the *until* operator $\mathbf{U}$ is defined on real trace languages by

$$L \,\mathbf{U}\, K = \{xy \mid x \in \mathbb{M},\ y \in K, \text{ and } x = x'x'',\ x'' \neq 1 \text{ implies } x''y \in L\}.$$

Remark 1. For comparison let us mention that the syntax and semantics of the logics LTrL defined in [17] and LTL_f defined in [2] are very similar. For LTrL, the difference is only that there is in addition for each letter $a \in \Sigma$ a constant $\langle a^{-1}\rangle\top$. Since the constant $\langle a^{-1}\rangle\top$ refers to the past, we need to use configurations to define its semantics: It holds $(z, x) \models \langle a^{-1}\rangle\top$ if and only if $a \in \max(x)$. For LTL_f, the difference is that for each subalphabet $B \subseteq \Sigma$ there is in addition a modality $\langle B^*\rangle\varphi$ whose semantics is given by $z \models \langle B^*\rangle\varphi$ if $z = xy$, $x \in \mathbb{M}_B$, and $y \models \varphi$. It is not clear whether there is a direct translation of LTrL or LTL_f to LTL; or between LTrL and LTL_f.

The following operators are standard abbreviations. They serve as macros.

$$\begin{array}{rll} \top := \neg\bot & & \textit{true}, \\ a := \langle a\rangle\top & & \text{for } a \in \Sigma, \\ A := \bigvee_{a\in A}\langle a\rangle\top & & \text{for } A \subseteq \Sigma, \\ \mathbf{X}\,\varphi := \bigvee_{a\in\Sigma}\langle a\rangle\varphi & & \textit{neXt } \varphi, \\ \mathrm{stop} := \neg\,\mathbf{X}\,\top & & \textit{termination}, \\ \mathbf{F}\,\varphi := \top\,\mathbf{U}\,\varphi & & \textit{future} \text{ or } \textit{eventually}\varphi, \\ \mathbf{G}\,\varphi := \neg\,\mathbf{F}\,\neg\varphi & & \textit{globally} \text{ or } \textit{always}\varphi. \end{array}$$

Example 1. $\mathrm{L}_{\mathbb{R}}(\mathrm{stop}) = \{1\}$, $\mathrm{L}_{\mathbb{R}}(\mathbf{F}\,\mathrm{stop}) = \mathbb{M}$ and for $A \subseteq \Sigma$ we have:

$$\begin{aligned}
\mathbb{R}_A &= \mathrm{L}_{\mathbb{R}}(\neg\,\mathbf{F}(\Sigma \setminus A)),\\
\mathbb{M}_A &= \mathrm{L}_{\mathbb{R}}(\mathbf{F}\,\mathrm{stop} \wedge \neg\,\mathbf{F}(\Sigma \setminus A)),\\
(\mathrm{Min} = A) &= \mathrm{L}_{\mathbb{R}}(\neg(\Sigma \setminus A) \wedge \textstyle\bigwedge_{a\in A} a),\\
(\mathrm{Max} \supseteq A) &= \mathrm{L}_{\mathbb{R}}(\textstyle\bigwedge_{a\in A} \mathbf{F}\langle a\rangle\mathrm{stop}),\\
(\mathrm{alphinf} = A) &= \mathrm{L}_{\mathbb{R}}(\mathbf{F}\,\mathbf{G}\,\neg(\Sigma \setminus A) \wedge \textstyle\bigwedge_{a\in A} \mathbf{G}\,\mathbf{F}\,a).
\end{aligned}$$

Remark 2. Later we shall perform an induction on the size of Σ leading to formulae $\varphi \in \mathrm{LTL}(A)$ for $A \subseteq \Sigma$. Such a formula may be interpreted over $\mathbb{R}_A$ or over $\mathbb{R}$ and we have $\mathrm{L}_{\mathbb{R}_A}(\varphi) = \mathrm{L}_{\mathbb{R}}(\varphi) \cap \mathbb{R}_A$. Hence $\mathrm{L}_{\mathbb{R}_A}(\varphi)$ is expressible in $\mathrm{LTL}(\Sigma)$. Another trivial observation is that if $\psi \in \mathrm{LTL}(\Sigma)$, then we can construct a formula $\psi_A \in \mathrm{LTL}(A)$ such that $\mathrm{L}_{\mathbb{R}}(\psi) \cap \mathbb{R}_A = \mathrm{L}_{\mathbb{R}_A}(\psi_A)$.

We will also use an induction on $|\Sigma|$ when the graph (Σ, D) is not connected. Then we find a partition of the alphabet $\Sigma = \Sigma_1 \cup \Sigma_2$ with $\Sigma_1 \times \Sigma_2 \subseteq I$ and each trace $x \in \mathbb{R}$ can be split into two independent traces x_1 and x_2 over Σ_1 and Σ_2 with $x = x_1x_2 = x_2x_1$. The following lemma can be shown by induction.

Lemma 1. *Assume that $\Sigma = \Sigma_1 \cup \Sigma_2$ with $\Sigma_1 \times \Sigma_2 \subseteq I$. Let $\mathbb{R}_i = \mathbb{R}_{\Sigma_i}$ and $\varphi_i \in \mathrm{LTL}(\Sigma_i)$ for $i = 1, 2$. Then $\mathrm{L}_{\mathbb{R}}(\varphi_1 \wedge \varphi_2) = \mathrm{L}_{\mathbb{R}_1}(\varphi_1) \cdot \mathrm{L}_{\mathbb{R}_2}(\varphi_2)$.*

4 First-Order Logic

The first order theory of traces is given by the syntax of $\mathrm{FO}(\Sigma, <)$:

$$\varphi ::= P_a(x) \mid x < y \mid \neg\varphi \mid \varphi \vee \varphi \mid (\exists x)\varphi,$$

where $a \in \Sigma$ and $x, y \in \mathrm{Var}$ are first order variables. Given a trace $t = [V, E, \lambda]$ and a valuation of the free variables into the vertices $\sigma : \mathrm{Var} \to V$, the semantics is obtained by interpreting the relation $<$ as the transitive closure of E and the predicate $P_a(x)$ by $\lambda(\sigma(x)) = a$. Then we can say when $(t, \sigma) \models \varphi$. If φ is a closed formula (a sentence), then the valuation σ has an empty domain and we define the language $\mathrm{L}_{\mathbb{R}}(\varphi) = \{t \in \mathbb{R} \mid t \models \varphi\}$. We say that a trace language $L \subseteq \mathbb{R}$ is expressible in $\mathrm{FO}(\Sigma, <)$ if there exists some sentence $\varphi \in \mathrm{FO}(\Sigma, <)$ such that $L = \mathrm{L}_{\mathbb{R}}(\varphi)$.

Passing from a temporal logic formula to a first order one is not very difficult. It is well-known or belongs to folklore. The transformation relies on the fact that a prefix (configuration) p of a trace t can be defined by its maximal vertices. Such a set of maximal vertices is bounded by the maximal number of pairwise independent letters in Σ. Therefore, a prefix inside a trace can be defined using a bounded number of first order variables.

Proposition 1. *If a trace language is expressible in* $\mathrm{LTL}(\Sigma)$*, then it is expressible in* $\mathrm{FO}(\Sigma, <)$*.*

5 Aperiodic Languages

Recall that a finite monoid S is *aperiodic*, if there is some $n \geq 0$ such that $s^n = s^{n+1}$ for all $s \in S$. A finitary trace language $L \subseteq \mathbb{M}$ is *aperiodic*, if there exists a morphism to some finite aperiodic monoid $h : \mathbb{M} \to S$ such that $L = h^{-1}(h(L))$. Since our considerations include infinite traces, we have to extend the notion of an aperiodic language to real trace languages such that it becomes equivalent for finitary languages.

Let $h : \mathbb{M} \to S$ be a morphism to some finite monoid S. For $x, y \in \mathbb{R}$, we say that x and y are h-similar, denoted by $x \sim_h y$ if either $x, y \in \mathbb{M}$ and $h(x) = h(y)$ or x and y have infinite factorizations in non-empty finite traces $x = x_1 x_2 \cdots$, $y = y_1 y_2 \cdots$ with $x_i, y_i \in \mathbb{M} \setminus \{1\}$ and $h(x_i) = h(y_i)$ for all i. According to the definition of h-similarity, we never have $x \sim_h y$ when x is finite and y is infinite. We denote by $\approx_h$ the transitive closure of $\sim_h$ which is therefore an equivalence relation. An equivalence class is denoted by $[x]_{\approx_h} = \{y \in \mathbb{R} \mid y \approx_h x\}$. For a finite trace $x \in \mathbb{M}$ we have $[x]_{\approx_h} = h^{-1}(h(x))$ and the monoid $\mathbb{M}$ is covered by at most $|S|$ classes. Using a Ramsey-type argument we can show that $\mathbb{R} \setminus \mathbb{M}$ is covered by at most $|S|^2$ classes. Therefore, $\{[x]_{\approx_h} \mid x \in \mathbb{R}\}$ defines a finite partition of $\mathbb{R}$ of cardinality at most $|S|^2 + |S|$. A real trace language $L \subseteq \mathbb{R}$ is *recognized* by h if it is saturated by $\sim_h$, i.e., $x \in L$ implies $[x]_{\approx_h} \subseteq L$ for all $x \in \mathbb{R}$. A real trace language $L \subseteq \mathbb{R}$ is *aperiodic* if it is recognized by some morphism to some finite and aperiodic monoid.

In order to prove the main theorem we shall use the equivalence between $\mathrm{FO}(\Sigma, <)$–definability and aperiodic languages.

Theorem 1 ([4,5]). *A finitary trace language (real trace language resp.) over an independence alphabet (Σ, I) is expressible in $\mathrm{FO}(\Sigma, <)$ if and only if it is aperiodic.*

We conclude this section by some easy results which will be useful later. The first proposition will allow us to use an induction on the size of the alphabet Σ.

Proposition 2. *Let $L \subseteq \mathbb{R}$ be a language recognized by the morphism $h : \mathbb{M} \to S$ into a finite monoid S and let $A \subseteq \Sigma$. Then, $L \cap \mathbb{R}_A$ and $L \cap \mathbb{M}_A$ are recognized by the restriction $h\restriction_{\mathbb{M}_A}$ of h to $\mathbb{M}_A$.*

Then, we consider the case where the dependence alphabet (Σ, D) is non-connected. A kind of Mezei's Theorem holds for real trace languages too:

Proposition 3. *Let $L \subseteq \mathbb{R}$ be a language recognized by the morphism $h : \mathbb{M} \to S$ into a finite monoid S. Assume that $\Sigma = \Sigma_1 \cup \Sigma_2$ with $\Sigma_1 \times \Sigma_2 \subseteq I$ and let $\mathbb{M}_i = \mathbb{M}_{\Sigma_i}$, $\mathbb{R}_i = \mathbb{R}_{\Sigma_i}$ and $h_i = h\restriction_{M_i}$ for $i = 1, 2$. Then, L is a finite union of products $L_1 \cdot L_2$ where $L_i \subseteq \mathbb{R}_i$ is recognized by h_i for $i = 1, 2$.*

Let $h : \mathbb{M} \to S$ be a morphism to some finite monoid S and let $L \subseteq \mathbb{R}$. For $s \in S$ we define $L(s) = \{x \in \mathbb{R} \mid h^{-1}(s)x \cap L \neq \emptyset\}$.

Proposition 4. *If L is recognized by h then $L = \bigcup_{s \in S} h^{-1}(s) \cdot L(s)$ and for each $s \in S$, the language $L(s)$ is recognized by h.*

6 Composition of LTL Languages

In the next section we will prove that each aperiodic trace language is expressible in LTL(Σ) using some induction. For this, we will write an aperiodic language as a finite union of products of simpler languages. By induction, the simpler languages will be shown to be expressible in LTL(Σ). Then we have to prove that their products and unions are still expressible in LTL(Σ). Since union corresponds with disjunction, the only problem is with concatenation. Though it is true in general that the product of languages expressible in LTL(Σ) is still expressible in LTL(Σ), we do not have a direct proof for this general result. Actually, this becomes a consequence of our main theorem since aperiodic trace languages are closed under product. In this section we show that some special products of expressible languages are still expressible in LTL(Σ). We state now two crucial composition lemmas. The proof techniques are similar, for lack of space we show the proof of Lemma 3 only.

Lemma 2. *Let $b \in \Sigma$, $B = \Sigma \setminus \{b\}$ and $\Pi = \mathbb{M}_B \cap (\mathrm{Max} \subseteq D(b))$. Let $L_1, L_2 \subseteq \mathbb{R}$ and $L_3 \subseteq (\mathrm{Min} = b)$ be trace languages expressible in* LTL(Σ)*. Then the language $(L_1 \cap \Pi)(L_2 \cap \mathbb{M}_{I(b)})L_3$ is expressible in* LTL(Σ)*.*

Lemma 3. *Let $b \in \Sigma$ and $B = \Sigma \setminus \{b\}$. Let $L_1 \subseteq \mathbb{R}$ and $L_2 \subseteq \mathbb{R}_B$ be expressible in* LTL(Σ)*. Then the language $(L_1 \cap (\mathrm{Max} \subseteq \{b\}))L_2$ is expressible in* LTL(Σ)*.*

Proof. The product $(\mathrm{Max} \subseteq \{b\})\mathbb{R}_B$ is unambiguous and we have

$$(L_1 \cap (\mathrm{Max} \subseteq \{b\}))L_2 = (L_1 \cap (\mathrm{Max} \subseteq \{b\}))\mathbb{R}_B \cap (\mathrm{Max} \subseteq \{b\})L_2.$$

1) We first show that $(L_1 \cap (\mathrm{Max} \subseteq \{b\}))\mathbb{R}_B$ is expressible in LTL(Σ). This is done by induction on the formula which defines L_1. The cases $\bot$ and $\varphi \vee \psi$ are trivial.

• $\neg\varphi$: Since the product $(\mathrm{Max} \subseteq \{b\})\mathbb{R}_B$ is unambiguous, we have

$$(\mathrm{L}_{\mathbb{R}}(\neg\varphi) \cap (\mathrm{Max} \subseteq \{b\}))\mathbb{R}_B = (\mathrm{Max} \subseteq \{b\})\mathbb{R}_B \setminus (\mathrm{L}_{\mathbb{R}}(\varphi) \cap (\mathrm{Max} \subseteq \{b\}))\mathbb{R}_B.$$

Moreover $(\mathrm{Max} \subseteq \{b\})\mathbb{R}_B$ is the set $(\mathrm{Alphinf} \subseteq B)$, which is expressible in LTL(Σ) by the formula $\mathbf{F}\,\mathbf{G}\,\neg b$.

• $\langle a\rangle\varphi$: The language $(\mathrm{L}_{\mathbb{R}}(\langle a\rangle\varphi) \cap (\mathrm{Max} \subseteq \{b\}))\mathbb{R}_B$ is equal to the language $a\Big((\mathrm{L}_{\mathbb{R}}(\varphi) \cap (\mathrm{Max} \subseteq \{b\}))\mathbb{R}_B \cap (a^{-1}(\mathrm{Max} = b))\mathbb{R}_B\Big)$. We conclude by induction since we have $(a^{-1}(\mathrm{Max} = b))\mathbb{R}_B = \mathrm{L}_{\mathbb{R}}(\psi)$ with

$$\psi = \bigvee_{a=b_0-b_1-\cdots-b_k=b} \mathbf{F}\langle b_1\rangle \cdots \mathbf{F}\langle b_k\rangle(\neg\,\mathbf{F}\,b)$$

where the disjunction ranges over all simple paths from a to b in the graph of the dependence alphabet (Σ, D).

• $\varphi\,\mathbf{U}\,\psi$: This case follows by induction from the formula

$$(\mathrm{L}_{\mathbb{R}}(\varphi\,\mathbf{U}\,\psi) \cap (\mathrm{Max} \subseteq \{b\}))\mathbb{R}_B = \\ \big(\mathrm{L}_{\mathbb{R}}(\varphi) \cap (\mathrm{Max} \subseteq \{b\})\big)\mathbb{R}_B\,\mathbf{U}\,\big(\mathrm{L}_{\mathbb{R}}(\psi) \cap (\mathrm{Max} \subseteq \{b\})\big)\mathbb{R}_B.$$

2) Next, if $\varphi \in \mathrm{LTL}(\Sigma)$ is a formula such that $\mathrm{L}_{\mathbb{R}}(\varphi) \subseteq \mathbb{R}_B$ then we have $(\mathrm{Max} \subseteq \{b\}) \cdot \mathrm{L}_{\mathbb{R}}(\varphi) = \mathrm{L}_{\mathbb{R}}((\mathbf{F}\, b)\, \mathbf{U}\, \varphi)$.

7 Kamp's Theorem for Real Traces

Theorem 2. *A finitary trace language (real trace language resp.) is expressible in* $\mathrm{FO}(\Sigma, <)$ *if and only if it is expressible in* $\mathrm{LTL}(\Sigma)$.

By Proposition 1 and Theorem 1 it is enough to show that all aperiodic languages in $\mathbb{R}(\Sigma, I)$ are expressible in $\mathrm{LTL}(\Sigma)$. The overall strategy is as in [2], but we must not use any filter operator. The filter operators allowed us to express directly a language of type $\mathbb{M}_A L(\varphi)$ for $A \subseteq \Sigma$ and $\varphi \in \mathrm{LTL}$. This simplified the proof considerably, but cannot be used here. We can circumvent these difficulties thanks to the results of the previous section.

Let Q be a finite set of states. We denote by $\mathrm{Trans}(Q)$ the monoid of mappings from Q to Q. The multiplication is the composition (in reverse order) of mappings: $(fg)(x) = g(f(x))$; and the unit element is the identity id_Q. We will use the fact that every finite monoid S can be realized as a submonoid of some $\mathrm{Trans}(Q)$ where $|Q| \leq |S|$. Indeed, it suffices to consider the right action of S over itself. More precisely, if we define $\chi(s) \in \mathrm{Trans}(S)$ by $\chi(s)(t) = ts$ then it is easy to see that $\chi : S \to \mathrm{Trans}(S)$ is an injective morphism.

We deduce that every aperiodic trace language can be recognized by some morphism $h : \mathbb{M}(\Sigma, I) \to S \subseteq \mathrm{Trans}(Q)$ where S is aperiodic. We show by induction on $(|Q|, |\Sigma|)$ that all languages recognized by h are expressible in $\mathrm{LTL}(\Sigma)$. We use the following well-founded lexicographic order on $\mathbb{N}^2$ for this induction: $(m, n) < (m', n')$ if and only if $m < m'$ or $m = m'$ and $n < n'$.

First, assume that $h(a) = \mathrm{id}_Q$ for all $a \in \Sigma$, which is in particular the case when $|Q| = 1$. If $L \subseteq \mathbb{R}$ is recognized by h then L is one of the sets $\emptyset$, $\mathbb{R}$, $\mathbb{M}$ or $\mathbb{R} \setminus \mathbb{M}$ which are respectively defined by the formulas $\bot$, $\top$, $\mathbf{F}\,\mathrm{stop}$, and $\mathbf{G}\,\mathbf{X}\,\top$. This shows the basis case of the induction.

Second, assume that $h(b) \neq \mathrm{id}_Q$ for some $b \in \Sigma$. The crucial observation here is that $h(b)$ is no permutation of Q. Indeed, since S is aperiodic, there exists some n such that $h(b)^n = h(b)^{n+1}$. If $h(b)$ were a permutation then it would be invertible and the last equality would imply $h(b) = \mathrm{id}_Q$, a contradiction. Hence, $h(b)(Q) = Q'$ for some $Q' \subset Q$ with $|Q'| < |Q|$.

We let $B = \Sigma \setminus \{b\}$ and we define two subsets of $\mathbb{M}$:

$$\Pi = \{x \in \mathbb{M}_B \mid \max(x) \subseteq D(b)\},$$
$$\Gamma = \{x \in \mathbb{M}_B \mid \min(x) \subseteq D(b)\}.$$

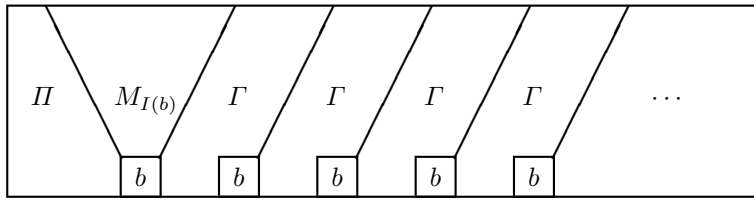
The notation Π is chosen since Πb are exactly the *pyramids* of $\mathbb{M}$ where the unique maximal element is b. It should be noted that $(\Pi b)^*$ and $(\Gamma b)^*$ are free submonoids of $\mathbb{M}$, both being infinitely generated if $D(b) \neq \{b\}$.

By Δ we denote the subset of real traces x which are either in $\mathbb{M}b$ or which can be factorized into an infinite product of *finite* traces such that all factors

except the first belong to Γb, that is, $\Delta = \mathbb{M}b(\Gamma b)^\infty$. The set Δ, which plays a key-role, admits the following unambiguous decomposition:

$$\Delta = \Pi \mathbb{M}_{I(b)} b(\Gamma b)^\infty.$$

This decomposition is best visualized by the following picture; it is in some sense the guide for the modular construction of a formula defining the language $L \cap \Delta$.



The core of the proof is now the following proposition.

Proposition 5. *Let $L \subseteq \mathbb{R}$ be recognized by h. Then, $L \cap \Delta$ is expressible in* $\mathrm{LTL}(\Sigma)$.

The proof of this proposition will be given later in Section 7.1. Here we show how it is used in the proof of Theorem 2. We start with two corollaries.

Corollary 1. *Assume that (Σ, D) is connected and let $L \subseteq \mathbb{R}$ be recognized by h. Then the language $L \cap (\mathrm{Alphinf} = \Sigma)$ is expressible in* $\mathrm{LTL}(\Sigma)$.

Proof. Since (Σ, D) is connected, we have $(\mathrm{Alphinf} = \Sigma) \subseteq \Delta$. Therefore,

$$L \cap (\mathrm{Alphinf} = \Sigma) = (L \cap \Delta) \cap (\mathrm{Alphinf} = \Sigma).$$

By Proposition 5 we know that $L \cap \Delta$ is expressible in $\mathrm{LTL}(\Sigma)$ and we conclude easily since $(\mathrm{Alphinf} = \Sigma) = \mathrm{L}_{\mathbb{R}}(\wedge_{a \in \Sigma} \mathbf{G}\,\mathbf{F}\, a)$.

Corollary 2. *Let $c \in \Sigma$, $C = \Sigma \setminus \{c\}$ and let $L \subseteq \mathbb{R}$ be recognized by h. Then the language $L \cap (\mathrm{Alphinf} \subseteq C)$ is expressible in* $\mathrm{LTL}(\Sigma)$.

Proof. We claim that $L \cap (\mathrm{Alphinf} \subseteq C)$ is the finite union

$$\bigcup_{u,v \in S} \left(\left((h^{-1}(u) \cap (\mathrm{Max} \subseteq \{b\})) \cdot (h^{-1}(v) \cap \mathbb{M}_B) \right) \cap (\mathrm{Max} \subseteq \{c\}) \right) \cdot (L(uv) \cap \mathbb{R}_C)$$

We will now apply Lemma 3 twice in order to conclude the proof of Corollary 2. We fix some $u, v \in S$. First, by Propositions 2 and 4 we know that $K_2 = L(uv) \cap \mathbb{R}_C$ is recognized by the morphism $h\restriction_{\mathbb{M}_C}$ and $L_2 = h^{-1}(v) \cap \mathbb{M}_B$ is recognized by the morphism $h\restriction_{\mathbb{M}_B}$. By induction on the size of the alphabet, we deduce that L_2 and K_2 are expressible in $\mathrm{LTL}(B)$ and $\mathrm{LTL}(C)$ respectively. By Remark 2 they are also expressible in $\mathrm{LTL}(\Sigma)$.

Second, note that $(\mathrm{Max} \subseteq \{b\}) \subseteq \Delta \cup \{1\}$. Hence,

$$h^{-1}(u) \cap (\mathrm{Max} \subseteq \{b\}) = (h^{-1}(u) \cap (\Delta \cup \{1\})) \cap (\mathrm{Max} \subseteq \{b\}).$$

By Proposition 5, the language $L_1 = h^{-1}(u) \cap (\Delta \cup \{1\})$ is expressible in $\mathrm{LTL}(\Sigma)$. Applying once Lemma 3 we deduce first that

$$K_1 = (L_1 \cap (\mathrm{Max} \subseteq \{b\})) \cdot L_2 = (h^{-1}(u) \cap (\mathrm{Max} \subseteq \{b\})) \cdot (h^{-1}(v) \cap \mathbb{M}_B)$$

is expressible in $\mathrm{LTL}(\Sigma)$. Next, applying a second time Lemma 3 we deduce that $(K_1 \cap (\mathrm{Max} \subseteq \{c\})) \cdot K_2$ is expressible in $\mathrm{LTL}(\Sigma)$. Therefore, $L \cap (\mathrm{Alphinf} \subseteq C)$ is expressible in $\mathrm{LTL}(\Sigma)$.

In order to conclude the proof of Theorem 2, we use the following proposition.

Proposition 6. *Assume that (Σ, D) is non-connected and let $L \subseteq \mathbb{R}$ be recognized by h. Then L is expressible in* $\mathrm{LTL}(\Sigma)$.

Proof. Assume that $\Sigma = \Sigma_1 \cup \Sigma_2$ with $\Sigma_1 \times \Sigma_2 \subseteq I$ and let $\mathbb{M}_i = \mathbb{M}_{\Sigma_i}$, $\mathbb{R}_i = \mathbb{R}_{\Sigma_i}$ and $h_i = h\restriction_{M_i}$ for $i = 1, 2$. By Proposition 3, L is a finite union of products $L_1 \cdot L_2$ where $L_i \subseteq \mathbb{R}_i$ is recognized by h_i for $i = 1, 2$. By induction on the size of the alphabet, we deduce that L_1 and L_2 are expressible in $\mathrm{LTL}(\Sigma_1)$ and $\mathrm{LTL}(\Sigma_2)$ respectively. Using Lemma 1, we deduce that $L_1 \cdot L_2$ is expressible in $\mathrm{LTL}(\Sigma)$.

7.1 Proof of Proposition 5

Let us first show that the set $(\Gamma b)^\infty$ is expressible in $\mathrm{LTL}(\Sigma)$.

Lemma 4. *The real trace language $(\Gamma b)^\infty$ is expressed by the formula*

$$(\mathrm{Min} \subseteq D(b)) \wedge \Big(\mathrm{stop} \vee \mathbf{F}\langle b\rangle \mathrm{stop} \vee \mathbf{G}\,\mathbf{F}(\mathrm{Min} = b)\Big)$$

where we write simply $(\mathrm{Min} \subseteq D(b))$ *and* $(\mathrm{Min} = b)$ *instead of the corresponding* $\mathrm{LTL}(\Sigma)$ *formula.*

Now, recall that $h(b)(Q) = Q'$. Each $s \in h((\Gamma b)^*)$ maps the subset Q' to Q'. Hence we may define two subsets $T, T' \subseteq \mathrm{Trans}(Q')$ by $T = \{s\restriction_{Q'} \mid s \in h(\Gamma b)\}$ and $T' = \{s\restriction_{Q'} \mid s \in h((\Gamma b)^*)\}$. Since $h((\Gamma b)^*)$ is a submonoid of S, the set T' is a monoid. Moreover, the monoid T' is generated by T and is aperiodic since S is aperiodic.

By T^* we denote the free monoid generated by the finite set T (here T is viewed as an alphabet). Accordingly, T^∞ means the set of finite or infinite words over the alphabet T. The inclusion $T \subseteq T'$ induces a canonical morphism $e : T^* \to T'$ which is called the *evaluation*.

The mapping $\sigma : \Gamma b \to T$, defined by the restriction $\sigma(x) = h(x)\restriction_{Q'}$, induces a morphism $\sigma : (\Gamma b)^* \to T^*$ between free monoids. The mapping σ is also

extended to infinite sequences $\sigma : (\Gamma b)^\omega \to T^\omega$, so finally we have $\sigma : (\Gamma b)^\infty \to T^\infty$.

Since T' is a submonoid of $\mathrm{Trans}(Q')$ and $|Q'| < |Q|$, we may use induction (Although we might have $|T| > |\Sigma|$). More precisely, we may assume that every language $K \subseteq T^\infty$ which is recognized by the morphism e is expressible in $\mathrm{LTL}(T)$. The following lemma allows us to make use of this induction step.

Lemma 5. *Let $K \subseteq T^\infty$ be a word language expressible in* $\mathrm{LTL}(T)$*. Then the real trace language $\sigma^{-1}(K) \subseteq (\Gamma b)^\infty$ is expressible in* $\mathrm{LTL}(\Sigma)$*.*

Proof. The statement of the lemma corresponds to Lemmas 5 and 9 in [2]. The proof of Lemma 5 given there can be translated to our situation without any difficulty. It is therefore omitted.

Lemma 6. *Let $L \subseteq \mathbb{R}$ be recognized by h. Then $L \cap b(\Gamma b)^\infty$ is expressible in* $\mathrm{LTL}(\Sigma)$*.*

Proof. We define the language $K \subseteq T^\infty$ with respect to the language L by

$$K = \{\sigma(x) \in T^\infty \mid bx \in L \cap b(\Gamma b)^\infty\}.$$

It was shown in [2, Lem. 6][1] that $L \cap b(\Gamma b)^\infty = b\sigma^{-1}(K)$ and that K is recognized by the morphism $e : T^* \to T'$.

Now, it is easy to conclude the proof. Since $|Q'| < |Q|$ and K is recognized by $e : T^* \to T' \subseteq \mathrm{Trans}(Q')$, we know by induction that K is expressible in $\mathrm{LTL}(T)$. Using Lemma 5, we deduce that $\sigma^{-1}(K)$ is expressible in $\mathrm{LTL}(\Sigma)$. Therefore, $L \cap b(\Gamma b)^\infty = b\sigma^{-1}(K)$ is expressible in $\mathrm{LTL}(\Sigma)$.

We are now ready to complete the proof of Proposition 5. Let $L \subseteq \mathbb{R}$ be a real trace language recognized by the morphism h. We claim that

$$L \cap \Delta = \bigcup_{u,v \in S} (h^{-1}(u) \cap \Pi)(h^{-1}(v) \cap \mathbb{M}_{I(b)})(L(uv) \cap b(\Gamma b)^\infty).$$

Indeed, let $t = xyz$ with $x \in h^{-1}(u)$, $y \in h^{-1}(v)$ and $z \in L(uv)$ for some $u, v \in S$. Then $h(xy) = uv$ and we deduce from Proposition 4 that $t = xyz \in L$. Conversely, let $t \in (L \cap \Delta)$. Using the unambiguous factorization $\Delta = \Pi \mathbb{M}_{I(b)} b(\Gamma b)^\infty$, we can write $t = xyz$ with $x \in \Pi$, $y \in \mathbb{M}_{I(b)}$ and $z \in b(\Gamma b)^\infty$. If we let $u = h(x)$ and $v = h(y)$ then we get $z \in L(uv)$ which concludes the proof of the claim.

By Proposition 4, we know that $L(uv)$ is recognized by h, hence by Lemma 6 the language $L_3 = L(uv) \cap b(\Gamma b)^\infty \subseteq (\mathrm{Min} = b)$ is expressible in $\mathrm{LTL}(\Sigma)$. By Proposition 2, the languages $L_1 = h^{-1}(u) \cap \mathbb{M}_B$ and $L_2 = h^{-1}(v) \cap \mathbb{M}_B$ are recognized by the restriction of h to $\mathbb{M}_B$. Hence, by induction on the size of the alphabet, they are expressible in $\mathrm{LTL}(B)$, and also in $\mathrm{LTL}(\Sigma)$ by Remark 2. Since

[1] In [2] we used a slightly different notion of h-similarity, but the proof given there applies in fact to the definition as it is used here.

$\Pi \cup \mathbb{M}_{I(b)} \subseteq \mathbb{M}_B$, we have $h^{-1}(u) \cap \Pi = L_1 \cap \Pi$ and $h^{-1}(v) \cap \mathbb{M}_{I(b)} = L_2 \cap \mathbb{M}_{I(b)}$. Using Lemma 2, we deduce that

$$(L_1 \cap \Pi)(L_2 \cap \mathbb{M}_{I(b)})L_3 = (h^{-1}(u) \cap \Pi)(h^{-1}(v) \cap \mathbb{M}_{I(b)})(L(uv) \cap b(\Gamma b)^\infty)$$

is expressible in LTL(Σ).

Acknowledgement

We thank Alexander Rabinovich for stimulating discussions during the Computer Science Logic conference CSL'99 at Madrid.

References

1. R. Alur, D. Peled, and W. Penczek. Model-checking of causality properties. In *Proceedings of LICS'95*, pages 90–100, 1995.
2. V. Diekert and P. Gastin. An expressively complete temporal logic without past tense operators for mazurkiewicz traces. In *Proceedings of CSL'99*, number 1683 in Lecture Notes in Computer Science, pages 188–203. Springer, 1999.
3. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
4. W. Ebinger. *Charakterisierung von Sprachklassen unendlicher Spuren durch Logiken*. Dissertation, Institut für Informatik, Universität Stuttgart, 1994.
5. W. Ebinger and A. Muscholl. Logical definability on infinite traces. *Theoretical Computer Science*, 154:67–84, 1996.
6. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Conference Record of the 12th ACM Symposium on Principles of Programming Languages*, pages 163–173, Las Vegas, Nev., 1980.
7. P. Gastin and A. Petit. Infinite traces. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 11, pages 393–486. World Scientific, Singapore, 1995.
8. J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, Calif., 1968.
9. R. Meyer and A. Petit. Expressive completeness of LTrL on finite traces: an algebraic proof. In *Proceedings of STACS'98*, number 1373 in Lecture Notes in Computer Science, pages 533–543, 1998.
10. M. Mukund and P. S. Thiagarajan. Linear time temporal logics over Mazurkiewicz traces. In *Proceedings of the 21th MFCS, 1996*, number 1113 in Lecture Notes in Computer Science, pages 62–92. Springer, 1996.
11. P. Niebert. A ν-calculus with local views for sequential agents. In *Proceedings of the 20th MFCS, 1995*, number 969 in Lecture Notes in Computer Science, pages 563–573. Springer, 1995.
12. W. Penczek. Temporal logics for trace systems: On automated verification. *International Journal of Foundations of Computer Science*, 4:31–67, 1993.
13. W. Penczek and R. Kuiper. Traces and logic. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, chapter 10, pages 307–390. World Scientific, Singapore, 1995.
14. R. Ramanujam. Locally linear time temporal logic. In *Proceedings of LICS'96*, Lecture Notes in Computer Science, pages 118–128, 1996.

15. P. S. Thiagarajan. A trace based extension of linear time temporal logic. In *Proceedings of LICS'94*, pages 438–447, 1994.
16. P. S. Thiagarajan. A trace consistent subset of PTL. In *Proceedings of CONCUR'95*, number 962 in Lecture Notes in Computer Science, pages 438–452, 1995.
17. P. S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. In *Proceedings of LICS'97*, 1997.
18. Th. Wilke. Classifying discrete temporal properties. In *Proceedings of STACS'99*, number 1563 in Lecture Notes in Computer Science, pages 32–46. Springer, 1999.

An Automata-Theoretic Completeness Proof for Interval Temporal Logic (Extended Abstract)

Ben C. Moszkowski*

Software Technology Research Lab.
SERCentre
Hawthorn Building
De Montfort University
The Gateway
Leicester LE1 9BH
Great Britain
benm@dmu.ac.uk
http://www.cms.dmu.ac.uk/~benm

Abstract. *Interval Temporal Logic* (ITL) is a formalism for reasoning about time periods. To date no one has proved completeness of a relatively simple ITL deductive system supporting infinite time and permitting infinite sequential iteration comparable to ω-regular expressions. We have developed a complete axiomatization for such a version of quantified ITL over finite domains and can show completeness by representing finite-state automata in ITL and then translating ITL formulas into them. Here we limit ourselves to finite time. The full paper (and another conference paper [15]) extends the approach to infinite time.

1 Introduction

Interval Temporal Logic (ITL) [6, 9, 10] is a temporal logic which includes a basic construct for the sequential composition of two formulas as well as an analog of Kleene star. Within ITL, one can express both finite-state automata and regular expressions. Its notation makes it suitable for logic-based modular reasoning involving periods of time, refinement [2], sequential composition using assumptions and commitments based on fixpoints of various temporal operators [12, 14] and for executable specifications [11]. Various imperative programming constructs are expressible in ITL and projection between time granularities is available (but not considered here). Zhou Chaochen, Hoare and Ravn [21] have developed an ITL extension called *Duration Calculus* for hybrid systems. Several researchers have looked at ITL decision procedures and axioms systems. However, previously there was no known complete axiom system for a version of ITL over both finite and ω-words having no artificial restrictions on interval constructs. We present a natural and complete axiomatization for a subset of quantified ITL

* Part of the research described here has been kindly supported by EPSRC research grant GR/K25922.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 223-234, 2000
© Springer-Verlag Berlin Heidelberg 2000

for finite time in which variables are limited to finite domains. The completeness proof describes an ITL decision procedure within ITL itself. In the full paper (and another conference paper [15]) infinite time is considered.

We build on the work of Siefkes [19] who proved the completeness of an axiomatization of the *Second-Order Theory of Successor* (S1S) and Kesten and Pnueli [7] who did likewise for *Quantified Propositional Temporal Logic* (QPTL) with past-time operators. Our approach follows Kesten and Pnueli's technique of reducing temporal formulas to finite-state automata as part of a decision procedure. These automata are themselves represented and manipulated in ITL. The ITL axiom system and completeness proof vary substantially from Kesten and Pnueli's. This reflects differences between conventional temporal logics and an interval-based one.

Our results offer a natural yet complete axiom system for a nontrivial subset of ITL and show how ITL can itself encode the decision procedure. Automata are more compositional than temporal logic tableaux which analyze several formulas in parallel. There is no need for Fischer-Ladner closures [4], first developed for a propositional version of Pratt's Dynamic Logic [17]. We also show that the ITL axiom system provides a logical framework for both finite-state automata and regular expressions.

2 Related Work

Let us now discuss other work on ITL axiom systems. Rosner and Pnueli [18] investigate an axiom system for quantifier-free propositional ITL (PITL) with finite and ω-intervals. The ITL subset also includes the *until* operator but not the operator *chop-star* which is like Kleene-star for regular expressions. A tableaux-based decision procedure underlies the completeness proof and uses an adaptation of the Fischer-Ladner closures. One inference rule requires detailed meta-reasoning about tableaux transitions.

Paech [16] investigates PITL with ω-intervals but includes a *chop-star* limited, like Kleene-star, to finitely many iterations and another operator *unless*. She gives a complete proof system with some nonconventional axioms for formulas already in a form like regular expressions and possibly involving complex meta-reasoning. A generalization of Fischer-Ladner closures is used.

Dutertre [3] gives two complete proof systems for first-order ITL without *chop-star* for finite time. One has a possible-worlds semantics of time and the other uses arbitrary linear orderings of states. Neither is complete for standard discrete-time intervals. Wang Hanpin and Xu Qiwen [20] generalize this to infinite time. Moszkowski [12] presents propositional and first-order ITL axiom systems for finite intervals. The former is claimed to be complete but only an outline of a proof is given. Axioms for temporal projection are given in [13]. Bowman and Thompson [1] have recently developed a tableaux-based completeness proof for an axiomatization of ITL with projection and finite time.

3 Overview of Interval Temporal Logic

We now briefly describe ITL for finite time. More details are in [6, 8–12, 14]. Basic ITL uses discrete, linear time. An interval σ has a length $|\sigma| \geq 0$ and a finite, nonempty

sequence of $|\sigma|+1$ states $\sigma_0, \ldots, \sigma_{|\sigma|}$. A state σ_i maps a variable such as A to a value $\sigma_i(A)$. Lower-case *static* variables $a, b, \ldots$ do not vary over time.

Here are permitted constructs using variable v, terms t and t' and formulas P and Q:

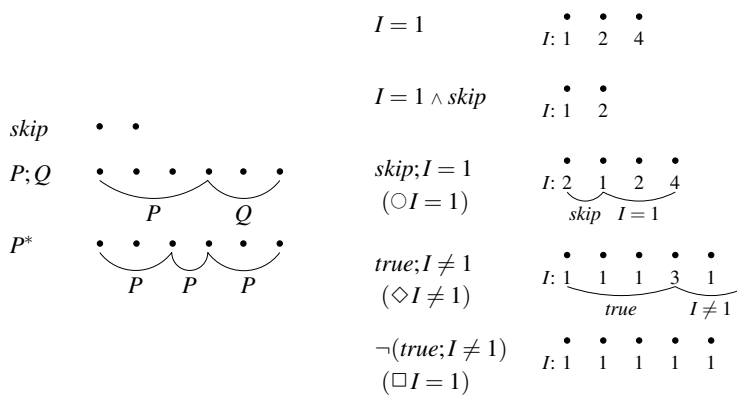
Terms: v (for numerical v), $0, 1, 2, \ldots$ (natural numbers), *if* P *then* t *else* t'
Formulas: v (for boolean v), $t = t'$, $\forall v.P$, $\neg P$, $P \wedge Q$, *skip*, $P;Q$, P^*

A variable v's values in an interval range over the finite, nonempty set $domain(v)$ which here is either $\{false, true\}$ or some initial subsequence of the natural numbers. Finite data domains ensure we have a decision procedure for our completeness result. We can readily extend *domain* to all ITL constructs. As in several temporal logics, the formula $I = 2$ is true on σ iff I's value in σ_0 equals 2.

There are three primitive temporal operators:

$$skip \qquad P;Q \ \ (chop) \qquad P^* \ \ (chop\text{-}star) \ ,$$

where P and Q are themselves formulas. The formula *skip* is true on a two-state interval. A formula $P;Q$ is true on σ iff σ can be chopped into two subintervals sharing a state σ_k for some $k \leq |\sigma|$ with P true on $\sigma_0 \ldots \sigma_k$ and Q true on $\sigma_k \ldots \sigma_{|\sigma|}$. Thus the formula $skip; I = J$ is true on σ iff σ has at least two states and $I = J$ is true in σ_1. A formula P^* is true on σ iff σ can be chopped into zero or more parts with P true on each. Any formula P^* (including $false^*$) is true on a one-state interval (see §3.2). Figure 1a pictorially illustrates the semantics of *skip*, *chop*, and *chop-star*. Some simple ITL formulas together with intervals which satisfy them are shown in Fig. 1b.



(a) Informal semantics (b) Some examples

Fig. 1. Informal ITL semantics and examples

For natural numbers i, j with $i \leq j \leq |\sigma|$, let $\sigma_{i:j}$ denotes the subinterval of length $j-i$ (i.e., $j-i+1$ states) with starting state σ_i and final state σ_j. Below is the syntax

and semantics of the basic ITL constructs used here. We denote the semantics of a term t and formula P on interval σ as $\mathcal{M}_\sigma[\![t]\!]$ and $\mathcal{M}_\sigma[\![P]\!]$.

3.1 Semantics of Terms

- Numerical static or state variable: $\mathcal{M}_\sigma[\![v]\!]=\sigma_0(v)$.
  A numerical variable's value for an interval σ is the value in σ's initial state σ_0.
- Numerical constant: $\mathcal{M}_\sigma[\![c]\!] = c$.
- Conditional term: $\mathcal{M}_\sigma[\![\mathit{if}\ P\ \mathit{then}\ t\ \mathit{else}\ t']\!] = \begin{cases} \mathcal{M}_\sigma[\![t]\!], & \text{if } \mathcal{M}_\sigma[\![P]\!] = \mathit{true} \\ \mathcal{M}_\sigma[\![t']\!], & \text{otherwise.} \end{cases}$

3.2 Semantics of Formulas

- Boolean static or state variable: $\mathcal{M}_\sigma[\![v]\!] = \sigma_0(v)$.
  A boolean variable's value for an interval σ is the value in σ's initial state σ_0.
- Equality: $\mathcal{M}_\sigma[\![t = t']\!] = \mathit{true} \quad \text{iff} \quad \mathcal{M}_\sigma[\![t]\!] = \mathcal{M}_\sigma[\![t']\!]$.
- Negation: $\mathcal{M}_\sigma[\![\neg P]\!] = \mathit{true} \quad \text{iff} \quad \mathcal{M}_\sigma[\![P]\!] = \mathit{false}$.
- Conjunction: $\mathcal{M}_\sigma[\![P \wedge Q]\!] = \mathit{true} \quad \text{iff} \quad \mathcal{M}_\sigma[\![P]\!] = \mathcal{M}_\sigma[\![Q]\!] = \mathit{true}$.
- Universal quantification: $\mathcal{M}_\sigma[\![\forall v.\, P]\!] = \mathit{true} \quad \text{iff} \quad \mathcal{M}_{\sigma'}[\![P]\!] = \mathit{true}$,
  for every interval σ' identical to σ except possibly for variable v's behavior.
- Unit interval: $\mathcal{M}_\sigma[\![\mathit{skip}]\!] = \mathit{true} \quad \text{iff} \quad |\sigma| = 1$.
- Chop: $\mathcal{M}_\sigma[\![P;Q]\!] = \mathit{true} \quad \text{iff} \quad \mathcal{M}_{\sigma'}[\![P]\!] = \mathit{true}$ and $\mathcal{M}_{\sigma''}[\![Q]\!] = \mathit{true}$,
  where $\sigma' = \sigma_{0:i}$ and $\sigma'' = \sigma_{i:|\sigma|}$ for some $i \leq |\sigma|$. Intervals σ' and σ'' share state σ_i.
- Chop-star: $\mathcal{M}_\sigma[\![P^*]\!] = \mathit{true} \quad \text{iff} \quad \mathcal{M}_{\sigma_{l_i:l_{i+1}}}[\![P]\!] = \mathit{true}$, for each $i : 0 \leq i < n$,
  for some $n \geq 0$ and finite sequence of natural numbers $l_0 \leq l_1 \leq \cdots \leq l_n$ where $l_0 = 0$ and $l_n = |\sigma|$. Every one-state interval satisfies P^* since we can trivially choose $n = 0$.

If a formula P is true on an interval σ, then σ *satisfies* P, denoted $\sigma \models P$. A formula P satisfied by all intervals is *valid*, denoted $\models P$.

We view formulas as boolean terms to avoid, for example, distinct theorems for quantified boolean and numerical variables. Hence $P = Q$ and $P \equiv Q$ are identical.

3.3 Some Definable Constructs

Constructs like *true*, $P \vee Q$ and $\exists v.\, P$ are definable as are $\Diamond P$ ("sometimes P"), $\Box P$ ("always P") and $\bigcirc P$ ("next P"):

$$\Diamond P \stackrel{\text{def}}{\equiv} \mathit{true}; P \qquad \Box P \stackrel{\text{def}}{\equiv} \neg \Diamond \neg P \qquad \bigcirc P \stackrel{\text{def}}{\equiv} \mathit{skip}; P \ .$$

We refer to the quantifier-free ITL subset built from no temporal operators but $\Diamond$ and $\bigcirc$ as *simple temporal logic*. Here are more operators expressible in this:

$ⓦ P$	$\stackrel{\text{def}}{\equiv} \neg \bigcirc \neg P$	(Weak next)	$\mathit{fin}\ P$	$\stackrel{\text{def}}{\equiv} \Box(\mathit{empty} \supset P)$	(Final state)
more	$\stackrel{\text{def}}{\equiv} \bigcirc \mathit{true}$	(More states)	$\mathit{halt}\ P$	$\stackrel{\text{def}}{\equiv} \Box(P \equiv \mathit{empty})$	(Just last)
empty	$\stackrel{\text{def}}{\equiv} \neg \mathit{more}$	(One state)	$ⓜ P$	$\stackrel{\text{def}}{\equiv} \Box(\mathit{more} \supset P)$	(Mostly)

The conventional temporal operator *until*, though definable (with $\exists$), is not needed. A version of $\bigcirc$ for numerical terms is expressible using conditional terms. The formula t *gets* t' is true iff for each pair of adjacent states, the value of term t' at the first state (i.e., on the suffix subinterval starting from it) equals term t's value at the second state:

$$t \; gets \; t' \quad \stackrel{\text{def}}{\equiv} \quad \boxed{\text{m}}\big((\bigcirc t) = t'\big) \quad .$$

Below are operators for examining *initial* and *arbitrary* subintervals:

$$\diamondsuit\!\!\!\!\cdot\; P \stackrel{\text{def}}{\equiv} P;true \qquad \boxed{\text{i}}\, P \stackrel{\text{def}}{\equiv} \neg \diamondsuit\!\!\!\!\cdot\; \neg P \qquad \diamondsuit\!\!\!\!\cdot\!\!\!\!\cdot\; P \stackrel{\text{def}}{\equiv} true;P;true \qquad \boxed{\text{a}}\, P \stackrel{\text{def}}{\equiv} \neg \diamondsuit\!\!\!\!\cdot\!\!\!\!\cdot\; \neg P \quad .$$

4 A Proof System

We now present a proof system for ITL. Our experience with hundreds of proofs has helped refine it. There is a quantifier-free part and another for quantifiers.

4.1 Quantifier-Free Axioms and Inference Rules.

We use some of Rosner and Pnueli's axioms for *chop* [18] and ours for $\boxed{\text{i}}$ and *chop-star* [12]. Let w be a *state formula*, i.e., without temporal operators.

Basic $\vdash$ Substitution instances of all valid quantifier-free state formulas.
P2 $\vdash (P;Q);R \equiv P;(Q;R)$
P3 $\vdash (P \vee P');Q \supset (P;Q) \vee (P';Q)$
P4 $\vdash P;(Q \vee Q') \supset (P;Q) \vee (P;Q')$
P5 $\vdash empty;P \equiv P$
P6 $\vdash P;empty \equiv P$
P7 $\vdash w \supset \boxed{\text{i}}\, w$
P8 $\vdash w \supset \Box w$, where variables in w are static.
P9 $\vdash \boxed{\text{i}}(P \supset P') \wedge \Box(Q \supset Q') \supset (P;Q) \supset (P';Q')$
P10 $\vdash \bigcirc P \supset \textcircled{\text{w}}\, P$
P11 $\vdash P \wedge \Box(P \supset \textcircled{\text{w}}\, P) \supset \Box P$
P12 $\vdash P^* \equiv empty \vee (P \wedge more);P^*$

MP $\vdash P \supset Q, \quad \vdash P \quad \Rightarrow \quad \vdash Q$
$\Box$**Gen** $\vdash P \quad \Rightarrow \quad \vdash \Box P$
$\boxed{\text{i}}$**Gen** $\vdash P \quad \Rightarrow \quad \vdash \boxed{\text{i}}\, P$

These axioms and inference rules do not have quantifiers but w, P, etc. can. In Axiom **Basic**, term t substitutes into variable v only if $domain(t) \subseteq domain(v)$. Axiom **P11** enables induction over time.

A formula P deduced from the axiom system is called an *ITL theorem*, denoted $\vdash P$. Below are a few theorems. The full paper has more with some proofs.

T1 $\vdash \Box(P \supset Q) \supset \Box P \supset \Box Q$
T2 $\vdash \bigcirc(P \supset Q) \supset \bigcirc P \supset \bigcirc Q$
T3 $\vdash \Diamond empty$
T4 $\vdash (w \wedge P);Q \quad \equiv \quad w \wedge (P;Q)$
T5 $\vdash P^{**} \equiv P^*$
T6 $\vdash skip^*$

4.2 Axioms and Inference Rules for Quantifiers

In the axioms and inference rules for quantifiers, v is an arbitrary variable:

Q1 $\vdash \forall v.\, P \quad \supset \quad P^t_v$,
where v is free for t in P and $domain\,(t) \subseteq domain\,(v)$. (The full paper describes substitution into temporal contexts.)

Q2 $\vdash \forall v.\,(P \supset Q) \;\supset\; (P \supset \forall v.\, Q)$, where v does not occur freely in P.
Q3 $\vdash \exists v.\,(P;Q) \quad \equiv \quad (\exists v.\, P);Q$, where v does not occur freely in Q.
Q4 $\vdash \exists v.\,(P;Q) \quad \equiv \quad P;(\exists v.\, Q)$, where v does not occur freely in P.
Q5 $\vdash (\exists v.\, P);\bigcirc(\exists v.\, Q) \;\supset\; \exists v.\,(P;\bigcirc Q)$, where v is a state variable.

$\forall$**Gen** $\vdash P \quad \Rightarrow \quad \vdash \forall v.\, P$, for any variable v.

The next theorem expresses *chop-star* using a fresh boolean variable B:

T7 $\vdash P^* \quad \equiv \quad \exists B.\Big(B \wedge \boxed{m}\big(B \;\supset\; \diamond\!\!\!\!\cdot(P \wedge \bigcirc halt\, B)\big)\Big)$.

Here is one to construct a hidden state variable v always equaling t:

T8 $\vdash \exists v.\, \Box(v = t)$,
where $domain\,(t) \subseteq domain\,(v)$ and v does not occur freely in t.

The one below creates a hidden state variable v which is initialized and then incrementally assigned a term which can depend on v's current value:

T9 $\vdash \exists v.\,(v = t \wedge v\; gets\; t')$,
where $domain\,(t) \subseteq domain\,(v)$ and $domain\,(t') \subseteq domain\,(v)$. Also v does not occur freely in t or within the temporal operators in t'.

Thus, the boolean variable B below initially equals *false* and always flips:

$$\vdash \quad \exists B.\,(B = false \wedge B\; gets\; \neg B) \;\; .$$

One can easily show that the axiom system is *sound*, that is, $\vdash P$ implies $\models P$. Our main goal is conversely to establish *completeness*, that is, $\models P$ implies $\vdash P$:

Theorem 4.1 (Completeness) *Any valid ITL formula is also a theorem.*

5 Overview of the Proof of Completeness

The basic completeness proof assumes formulas contain no static variables:

Lemma 5.1 (Relative completeness for static variables) *If all valid formulas without static variables are theorems, so are those with them.*

Proof (Outline) Let P be a valid formula and let P' be $\forall u_1 \ldots \forall u_n.\, P$, where $u_1, \ldots, u_n$ are the free static variables in P. Now P' is also valid and provably implies P (i.e., $\vdash P' \supset P$). For each static variable u in P', replace any subformula $\forall u.\, Q$ using the theorem $\vdash \forall u.\, Q \equiv \bigwedge_{c \in domain\,(u)} Q^c_u$. The new formula P'' is valid and provably equivalent to P' (i.e., $\vdash P' \equiv P''$). By our assumption, P'' is a theorem so we can deduce P. $\square$

The proof of Theorem 4.1 reduces formulas to equivalent ones in a *normal form*:

Lemma 5.2 (Normal form) *For each formula we can deduce an equivalent one having no new variables and in which each equality is of the form $v = c$, where v is numerical and $c \in domain(v)$. If the original formula is the simple temporal logic defined in §3.3, so is the normalized one.*

Lemma 5.3 (Completeness for simple temporal logic) *Any valid formula in the simple temporal logic subset is a theorem.*

Proof (Outline) We start with a complete axiom system for conventional linear-time temporal logic easily altered for finite intervals and finite domains. All of the axioms and inference rules are provable from our ITL axiom system. They are Axioms **Basic**, **P8**, **P10** and **P11**, Inference Rules **MP** and □**Gen** and Theorems **T1**, **T2** and **T3**. Therefore, ITL theoremhood of any valid formula in the subset can be deduced. □

5.1 Automata

We now adapt the approach of Kesten and Pnueli [7] and utilize a variant of finite-state automata called here *chop-automata* which selectively accept intervals. All normalized ITL formulas can be built from a few constructs, each having a translation into an automaton. In effect, we embed a decision procedure for ITL in ITL itself and use the logic to express the procedure's correctness. This helps to show completeness.

The behavior of an automaton $\mathcal{A}$ can be expressed in ITL as the formula denoted $\chi^{\mathcal{A}}$ (defined later) which is true on an interval iff $\mathcal{A}$ accepts that interval. We then construct from a formula P an automaton $\mathcal{A}^P$ accepting the intervals satisfying P. The formula $\chi^{\mathcal{A}^P}$ represents $\mathcal{A}^P$'s accepting runs and is provably equivalent to P in the axiom system (i.e., $\vdash P \equiv \chi^{\mathcal{A}^P}$). The shorter form χ^P is generally used. We can also show for any automata $\mathcal{A}$ accepting no intervals, the formula $\chi^{\mathcal{A}}$ is provably false (i.e., $\vdash \neg\chi^{\mathcal{A}}$).

To prove that a valid formula P is a theorem, we construct from $\neg P$ an automaton $\mathcal{A}^{\neg P}$ with $\vdash \neg P \equiv \chi^{\neg P}$. Now P is valid, so $\mathcal{A}^{\neg P}$ accepts nothing and we deduce $\vdash \neg\chi^{\neg P}$. These together yield $\vdash P$.

We now describe *chop-automata* for recognizing finite intervals.

Definition 5.4 (Chop-automaton) *A (nondeterministic)* chop-automaton $\mathcal{A}$ *is a quintuple* $(V, K, q_0, \delta, \tau)$ *for which*

- *V is a possibly empty finite set of boolean and numerical state variables,*
- *K is a nonempty finite set of* automaton states*,*
- *$q_0 \in K$ is the* initial state*,*
- *δ is the* transition function *mapping $K \times K$ to quantifier-free state formulas over variables in V,*
- *τ is the* termination function *mapping K to quantifier-free state formulas over variables in V.*

It is necessary to introduce the notion of a *run* of a chop-automaton on an interval:

Definition 5.5 (Run and accepting run of chop-automaton) *A* run *of a chop-automata $\mathcal{A}$ over an interval σ is any finite sequence ρ of $|\sigma|+1$ elements $\rho_0, \ldots, \rho_{|\sigma|} \in K$ in which for each two adjacent automaton states ρ_i and ρ_{i+1} the interval state σ_i satisfies the transition formula $\delta(\rho_i, \rho_{i+1})$, (i.e., $\sigma_i \models \delta(\rho_i, \rho_{i+1})$).*

A run ρ is called an accepting run *of the chop-automaton $\mathcal{A}$ over the interval σ if the run's initial state ρ_0 is q_0 and in addition σ's final state $\sigma_{|\sigma|}$ satisfies the termination condition selected by the run's final automaton state $\rho_{|\sigma|}$, namely $\tau(\rho_{|\sigma|})$.*

We say that $\mathcal{A}$ accepts *an interval σ if there is at least one accepting run over σ.*

In contrast to conventional finite-state automata, a chop-automaton uses τ to test the very end of an interval without advancing to permit the operator *chop* to be represented.

An automaton $\mathcal{A}$'s accepting runs are expressible in ITL. Let Y be a numerical state variable *not* in V and with $K \subseteq domain(Y)$. Define the formula $acc_r^{\mathcal{A}}(Y)$ as follows:

$$acc_r^{\mathcal{A}}(Y) \quad \stackrel{\text{def}}{\equiv} \quad Y = q_0 \wedge \boxed{\text{m}}\, \delta(Y, \bigcirc Y) \wedge fin\, \tau(Y) \ .$$

The formula $\chi^{\mathcal{A}}$ now defined expresses the existence of some accepting run:

$$\chi^{\mathcal{A}} \quad \stackrel{\text{def}}{\equiv} \quad \exists Y.\, acc_r^{\mathcal{A}}(Y) \ .$$

5.2 Automata Constructions

Given some normalized formula P (see Lemma 5.2 presented earlier), we construct an automaton $\mathcal{A}^P$. We sometimes denote the individual parts of $\mathcal{A}^P$ as V^P, K^P, etc. and abbreviate $acc_r^{\mathcal{A}^P}(Y)$ and $\chi^{\mathcal{A}^P}$ as $acc_r^P(Y)$ and χ^P, respectively.

Here is a list of formulas which we need to consider: w (quantifier-free state formula), $P \vee Q$, $\neg P$, $\exists v.\, P$, *skip*, and $P; Q$. These constructs are ones most readily translated to automata. We replace *chop-star* formulas using ITL Theorem **T7** in §4.2 to avoid the need to also directly reduce them to automata.

During the construction of automata, various operations can be performed such as renaming of an automaton's states or determinizing it. These are expressible as ITL theorems. The details are omitted here. We also have the following lemma:

Lemma 5.6 *If $\mathcal{A}$ has no accepting runs, then $\vdash \neg\chi^{\mathcal{A}}$.*

Proof Suppose that $\mathcal{A}$ has no accepting runs. The following formula is valid and hence a theorem by Lemma 5.3: $\vdash \neg acc_r^{\mathcal{A}}(Y)$. We introduce an existential quantifier to deduce the theorem $\vdash \neg\exists Y.\, acc_r^{\mathcal{A}}(Y)$ which reduces to $\vdash \neg\chi^{\mathcal{A}}$. □

Below are constructions for w, *skip* and *chop*. The full paper also looks at others.

Automata for Quantifier-Free State Formulas For a quantifier-free state formula w, the automaton $\mathcal{A}^w$ has V equal the set of w's variables, $K = \{0,1\}$, $q_0 = 0$ with δ and τ as follows:

$$\begin{array}{llll} \delta(0,0): \mathit{false} & \delta(0,1): w & \tau(0): w & \tau(1): \mathit{true} \\ \delta(1,0): \mathit{false} & \delta(1,1): \mathit{true} & & \end{array}$$

Claim 5.7 *The automaton $\mathcal{A}^w$ accepts an interval σ iff σ satisfies w.*

Lemma 5.8 *The following equivalence is an ITL theorem:* $\vdash w \equiv \chi^w$.

Proof Let X be a numerical state variable with domain $\{0,1\}$ and not occurring in w. The following valid simple temporal formula is a theorem by Lemma 5.3:

$$\vdash \quad w \wedge X = 0 \wedge X \; gets \; 1 \;\supset\; acc_r^w(X) \;\; . \tag{5.1}$$

The variable X can now be existentially created using ITL Theorem **T9**:

$$\vdash \quad \exists X.\,(X = 0 \wedge X \; gets \; 1) \;\; . \tag{5.2}$$

The two theorems (5.1) and (5.2) are combined to obtain the following:

$$\vdash \quad w \;\supset\; \exists X.\,acc_r^w(X) \;\; . \tag{5.3}$$

For the converse, the valid formula $acc_r^w(X) \supset w$ is a theorem by Lemma 5.3. We then deduce $\vdash \exists X.\,acc_r^w(X) \supset w$ which with (5.3) yields $\vdash w \equiv \chi^w$. □

Automaton for *skip* Below is an automaton $\mathcal{A}^{skip}$ accepting two-state intervals:

$$V = \{\},\; K = \{0,1\},\; q_0 = 0,$$

$$\begin{array}{llll} \delta(0,0)\colon false & \delta(0,1)\colon true & \tau(0)\colon false & \tau(1)\colon true \\ \delta(1,0)\colon false & \delta(1,1)\colon false & & \end{array}$$

Claim 5.9 *The automaton $\mathcal{A}^{skip}$ accepts an interval σ iff σ satisfies skip.*

Lemma 5.10 *The following equivalence is provably true:* $\vdash \; skip \equiv \chi^{skip}$.

Proof We first look at deducing $skip \;\supset\; \chi^{skip}$. Let X have domain $\{0,1\}$. The next valid formula is a theorem by Lemma 5.3:

$$\vdash \quad skip \wedge \Box(X = if \; more \; then \; 0 \; else \; 1) \;\supset\; acc_r^{skip}(X) \;\; . \tag{5.4}$$

A hidden instance of X is now created with ITL Theorem **T8** in §4.2:

$$\vdash \quad \exists X.\,\Box(X = if \; more \; then \; 0 \; else \; 1) \;\; . \tag{5.5}$$

We then combine the two theorems (5.4) and (5.5):

$$\vdash \quad skip \;\supset\; \exists X.\,acc_r^{skip}(X) \;\; . \tag{5.6}$$

For the converse, the valid formula below is a theorem by Lemma 5.3:

$$\vdash \quad acc_r^{skip}(X) \;\supset\; skip \;\; .$$

The variable X is existentially hidden to deduce the following:

$$\vdash \quad \exists X.\,acc_r^{skip}(X) \;\supset\; skip \;\; . \tag{5.7}$$

We reach the goal by combining (5.6) and (5.7): $\vdash skip \equiv \chi^{skip}$. □

Automata for *chop* Let us now construct an automaton $\mathcal{A}^{P;Q}$ for the formula $P;Q$ and deduce the equivalence of $P;Q$ and $\chi^{P;Q}$. Assume by induction that $\mathcal{A}^P$ and $\mathcal{A}^Q$ are P's and Q's respective automata with disjoint K^P and K^Q. Here is a suitable $\mathcal{A}^{P;Q}$:

$$V = V^P \cup V^Q,\ K = K^P \cup K^Q,\ q_0 = q_0^P,$$

$$\begin{array}{lll} & \delta(q,q'): & \tau(q): \\ \delta^P(q,q'), & \text{for } q,q' \in K^P & \tau^P(q) \wedge \tau^Q(q_0^Q), \ \text{for } q \in K^P \\ \delta^Q(q,q'), & \text{for } q,q' \in K^Q & \tau^Q(q) \quad\quad \text{for } q \in K^Q \\ \tau^P(q) \wedge \delta^Q(q_0^Q,q'), & \text{for } q \in K^P, q' \in K^Q & \\ \mathit{false}, & \text{otherwise} & \end{array}$$

Claim 5.11 *The automaton $\mathcal{A}^{P;Q}$ accepts an interval σ iff σ satisfies $P;Q$.*

Lemma 5.12 *Formulas $P;Q$ and $\chi^{P;Q}$ are provably equivalent: $\vdash P;Q \equiv \chi^{P;Q}$.*

Proof We assume by induction $\vdash P \equiv \chi^P$ and $\vdash Q \equiv \chi^Q$ and then deduce the following:

$$\vdash P;Q \equiv \chi^P;\chi^Q \tag{5.8}$$

To show that the valid formula $\chi^P;\chi^Q \equiv \chi^{P;Q}$ is a theorem, we re-express it:

$$\big(\exists X.\, acc_r^P(X)\big);\big(\exists Y.\, acc_r^Q(Y)\big) \quad\equiv\quad \exists Z.\, acc_r^{P;Q}(Z)\ .$$

Here X, Y and Z share a domain which is a superset of K^P, K^Q and K. The left subformula's quantifiers can be moved out of the *chop* operator:

$$\vdash\ \big(\exists X.\, acc_r^P(X)\big);\big(\exists Y.\, acc_r^Q(Y)\big) \equiv \exists X,Y.\, \big(acc_r^P(X);acc_r^Q(Y)\big)\ . \tag{5.9}$$

We now turn to the formula $acc_r^P(X);acc_r^Q(Y)$. This is provably equivalent to the formula $\exists B.\, \phi(B,X,Y)$, where B is a new boolean state variable and the subformula $\phi(B,X,Y)$ is as now defined:

$$\begin{array}{lll} \phi(B,X,Y) & \stackrel{\text{def}}{\equiv} & X = q_0^P \wedge B \\ & & \wedge\ \boxed{\mathrm{m}}(B \supset \delta^P(X,\bigcirc X)) \\ & & \wedge\ \boxed{\mathrm{m}}(\neg B \supset \delta^Q(Y,\bigcirc Y) \wedge \bigcirc\neg B) \\ & & \wedge\ \Box\big(B \wedge \textcircled{w}\neg B \supset \tau^P(X) \wedge Y = q_0^Q \wedge \big(\mathit{empty} \vee \delta^Q(Y,\bigcirc Y)\big)\big) \\ & & \wedge\ \mathit{fin}\ \tau^Q(Y)\ . \end{array}$$

We represent the automata's behavior using ϕ because it is in simple temporal logic. The purpose of B is to indicate at each interval state which of the *chop* construct's two subintervals contains the state. When B is true, the state is enclosed in the left subinterval and automaton $\mathcal{A}^P$ is active. When B is false the state is within the right one and $\mathcal{A}^Q$ is active. In the case of the single state shared by both intervals, B remains true as in the left subinterval. If the right subinterval has only one state then B is always true.

The relationship between ϕ and $acc_r^P(X);acc_r^Q(Y)$ is now expressed:

$$\vdash\quad \exists B.\, \phi(B,X,Y) \quad\equiv\quad acc_r^P(X);acc_r^Q(Y)\ . \tag{5.10}$$

In order to prove this, we first deduce the next theorem:

$$\vdash \quad \phi(B,X,Y) \quad \equiv \quad (acc_r^P(X) \wedge \Box B);(acc_r^Q(Y) \wedge B \wedge ⓦ\Box\neg B) \ .$$

Its proof uses temporal fixpoints and derived inference rules for compositionality. The details are omitted here. Lemma 5.3 yields the theoremhood of some valid formulas involving ϕ such as the next one for creating Z from B, X and Y:

$$\vdash \quad \phi(B,X,Y) \wedge \Box(Z = \mathit{if}\ B\ \mathit{then}\ X\ \mathit{else}\ Y) \ \supset\ acc_r^{P;Q}(Z) \ .$$

We combine this with ITL Theorem **T8** in §4.2 and hide B and Z:

$$\vdash \quad \exists B.\phi(B,X,Y) \quad \supset \quad \exists Z.acc_r^{P;Q}(Z) \ .$$

From this and ITL Theorems (5.9) and (5.10) we achieve half of the goal:

$$\vdash \quad \big(\exists X.acc_r^P(X)\big);\big(\exists Y.acc_r^Q(Y)\big) \ \supset\ \exists Z.acc_r^{P;Q}(Z) \ . \tag{5.11}$$

For the converse of (5.11), we first deduce the next valid formula using Lemma 5.3:

$$\begin{aligned}\vdash \quad & acc_r^{P;Q}(Z) \wedge \Box\big(B \equiv (Z \in K^P)\big) \wedge \Box(X = Z) \\ & \wedge Y = q_0^Q \wedge Y\ \mathit{gets}\ \big(\mathit{if}\ (\bigcirc B)\ \mathit{then}\ Y\ \mathit{else}\ \bigcirc Z\big) \\ & \quad \supset \quad \phi(B,X,Y) \ .\end{aligned}$$

We then existentially hide B, X and Y using ITL Theorems **T8** and **T9**:

$$\vdash \quad acc_r^{P;Q}(Z) \ \supset\ \exists B,X,Y.\phi(B,X,Y) \ .$$

This, (5.9) and (5.10) lead to our desired equivalence's other direction:

$$\vdash \quad \exists Z.acc_r^{P;Q}(Z) \ \supset\ \big(\exists X.acc_r^P(X)\big);\big(\exists Y.acc_r^Q(Y)\big) \ . \tag{5.12}$$

From ITL Theorems (5.11) and (5.12) and χ's definition, we get $\vdash \chi^P;\chi^Q \ \equiv\ \chi^{P;Q}$. This and theorem (5.8) yield our main goal: $\vdash P;Q \equiv \chi^{P;Q}$. □

6 Discussion

The version of ITL here uses numerical variables. Alternatively, we can restrict all variables to being boolean and encode numbers as Kesten and Pnueli do.

In [13] we look at a compositional axiom system for temporal projection over finite intervals which is claimed to be complete. We would also like support for ω-intervals.

Hale [5] first studied *framing* in ITL. At present, if a state variable does not change value, this must be explicitly specified. Framing makes this implicit and shortens specifications. A complete axiom system for framing would be helpful.

Acknowledgments

The author thanks Antonio Cau and Jordan Dimitrov for suggesting improvements to the presentation. Moshe Vardi and Wolfgang Thomas also provided helpful advice.

References

[1] H. Bowman and S. J. Thompson. A complete axiomatization of Interval Temporal Logic with projection. Technical Report 6-00, Computing Lab., Univ. of Kent, UK, Jan. 2000.

[2] A. Cau and H. Zedan. Refining Interval Temporal Logic specifications. In M. Bertran and T. Rus, eds., *Transformation-Based Reactive Systems Development*, LNCS 1231, pp. 79–94. AMAST, Springer-Verlag, 1997.

[3] B. Dutertre. Complete proof systems for first order interval temporal logic. In *Proc. 10th LICS*, pp. 36–43. IEEE Computer Soc. Press, June 1995.

[4] M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, Apr. 1979.

[5] R. W. S. Hale. *Programming in Temporal Logic*. PhD thesis, Computer Laboratory, Cambridge University, Cambridge, England, Oct. 1988.

[6] J. Halpern, Z. Manna, and B. Moszkowski. A hardware semantics based on temporal intervals. In J. Diaz, ed., *Proc. ICALP '83*, LNCS 154, pp. 278–291. Springer-Verlag, 1983.

[7] Y. Kesten and A. Pnueli. A complete proof system for QPTL. In *Proc. 10th LICS*, pp. 2–12. IEEE Computer Soc. Press, 1995.

[8] B. Moszkowski. *Reasoning about Digital Circuits*. PhD thesis, Dept. Computer Science, Stanford Univ., 1983. Tech. rep. STAN–CS–83–970.

[9] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. In *Proc. 6-th Int'l. Symp. on Computer Hardware Description Languages*, pp. 79–90, Pittsburgh, Penn., May 1983. North-Holland Pub. Co.

[10] B. Moszkowski. A temporal logic for multilevel reasoning about hardware. *Computer*, 18:10–19, 1985.

[11] B. Moszkowski. *Executing Temporal Logic Programs*. Cambridge U. Press, 1986.

[12] B. Moszkowski. Some very compositional temporal properties. In E.-R. Olderog, ed., *Programming Concepts, Methods and Calculi*, IFIP Transactions A-56, pp. 307–326. IFIP, Elsevier Science B.V. (North–Holland), 1994.

[13] B. Moszkowski. Compositional reasoning about projected and infinite time. In *Proc. 1st IEEE Int'l Conf. on Engineering of Complex Computer Systems (ICECCS'95)*, pp. 238–245. IEEE Computer Soc. Press, 1995.

[14] B. Moszkowski. Compositional reasoning using Interval Temporal Logic and Tempura. In W.-P. de Roever et al., eds., *Compositionality: The Significant Difference*, LNCS 1536, pp. 439–464. Springer-Verlag, 1998.

[15] B. Moszkowski. A complete axiomatization of interval temporal logic with infinite time. In *Proc. 15th Annual IEEE Symp. on Logic in Computer Science (LICS 2000)*. IEEE Computer Soc. Press, June 2000.

[16] B. Paech. Gentzen-systems for propositional temporal logics. In E. Börger et al., eds., *Proc. 2nd Workshop on Computer Science Logic, Duisburg (FRG)*, LNCS 385, pp. 240–253. Springer-Verlag, Oct. 1988.

[17] V. R. Pratt. Semantical considerations on Floyd-Hoare logic. In *17-th Annual IEEE Symposium on Foundations of Computer Science*, pp. 109–121, 1976.

[18] R. Rosner and A. Pnueli. A choppy logic. In *Proc. 1st LICS*, pp. 306–313. IEEE Computer Soc. Press, June 1986.

[19] D. Siefkes. *Decidable Theories I: Büchi's Monadic Second Order Successor Arithmetic*, Lecture Notes in Mathematics 120. Springer-Verlag, Berlin, 1970.

[20] Wang Hanpin and Xu Qiwen. Temporal logics over infinite intervals. Technical Report 158, UNU/IIST, Macau, 1999.

[21] Zhou Chaochen, C. A. R. Hoare, and A. P. Ravn. A calculus of durations. *Inf. Process. Lett.*, 40(5):269–276, 1991.

Which NP-Hard Optimization Problems Admit Non-trivial Efficient Approximation Algorithms?

J. Håstad

Royal Institute of Technology
Sweden
johanh@nada.kth.se

Abstract. In 1991 Feige, Goldwasser, Lovász, Safra and Szegedy found a connection between the approximability of Max-Clique and the area of multiprover interactive proofs. What first seemed like an isolated result have developed into an entire area of research.
The connection between interactive proofs and in particular the variant called probabilistically checkable proofs or PCPs and inapproximability results for many NP-hard optimization problems has proved to be fundamental. For some optimization problems, like clique and many constraint satisfaction problems, the parameter giving the degree of inapproximability corresponds to a natural parameter in the PCP. Other times, like for colorability and the traveling salesman problem the correspondence is not as direct, but the best results are still derived from PCPs.
The basic qualitative result on the existence of efficient PCPs was given in the famous paper by Arora, Lund, Motwani, Sudan and Szegedy in 1992 but since then the quantitative results have improved considerably. For many problems we now have almost tight results while for some others our knowledge is less complete.
The goal of this talk is to give an understanding of what has happened, outline a few of the results and give at least a taste of some ingredients of the proofs involved.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, p. 235, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Deterministic Algorithms for k-SAT Based on Covering Codes and Local Search

Evgeny Dantsin[1,⋆], Andreas Goerdt[2],
Edward A. Hirsch[3,⋆⋆], and Uwe Schöning[4]

[1] Steklov Institute of Mathematics, 27 Fontanka, 191011 St.Petersburg, Russia,
dantsin@pdmi.ras.ru
[2] TU Chemnitz, Fakultät für Informatik, 09107 Chemnitz, Germany,
goerdt@informatik.tu-chemnitz.de
[3] Steklov Institute of Mathematics, 27 Fontanka, 191011 St.Petersburg, Russia,
hirsch@pdmi.ras.ru, http://logic.pdmi.ras.ru/~hirsch
[4] Universität Ulm, Abteilung Theoretische Informatik, 89069 Ulm, Germany,
schoenin@informatik.uni-ulm.de

Abstract. We show that satisfiability of formulas in k-CNF can be decided deterministically in time close to $(2k/(k+1))^n$, where n is the number of variables in the input formula. This is the best known worst-case upper bound for deterministic k-SAT algorithms. Our algorithm can be viewed as a derandomized version of Schöning's probabilistic algorithm presented in [15]. The key point of our algorithm is the use of covering codes together with local search. Compared to other "weakly exponential" algorithms, our algorithm is technically quite simple.
We also show how to improve the bound above by moderate technical effort. For 3-SAT the improved bound is 1.481^n.

1 Introduction

Worst-Case Upper Bounds for SAT. The satisfiability problem for propositional formulas (SAT) can be decided by an obvious algorithm in time[1] 2^n, where n is the number of variables in the input formula. This worst-case upper bound can be decreased for k-SAT, i.e., if we restrict inputs to formulas in conjunctive normal form with at most k literals per clause (k-CNF). First upper bounds c^n, where $c < 2$, were obtained in 1980s [4,10], for example, the bound 1.619^n for 3-SAT. Currently, much research in SAT algorithms is aimed at decreasing the base in exponential upper bounds, e.g., [16,14,8,13,5,12,7,15].

The best known bound for probabilistic 3-SAT algorithms is $(4/3)^n$ due to U. Schöning [15]. For probabilistic k-SAT algorithms when $k \geq 4$, the best known bound is due to R. Paturi, P. Pudlák, M. E. Saks, and F. Zane [12]. This bound

⋆ Work done in part while visiting Department of Computer Science of University of Manchester. Supported by grants from TFR, INTAS, and RFBR.
⋆⋆ Supported by INTAS project 96-0760 and RFBR grant 99-01-00113.
[1] In this paper, we write all complexity bounds up to a polynomial in the size of the input formula. For example, we write 2^n instead of $poly(n)2^n$.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 236–247, 2000.
© Springer-Verlag Berlin Heidelberg 2000

is not represented in compact form; the bound for 4-SAT is 1.477^n, the bound for 5-SAT is 1.569^n.

The best known bounds for deterministic k-SAT algorithms are as follows. For $k = 3$, O. Kullmann [7] gives the bound 1.505^n. In [14], the bound 1.497^n was announced ([6] sketched how this bound can be obtained by a refinement of [7]). For $k = 4$, the best known bound is still 1.840^n due to B. Monien and E. Speckenmeyer [10]. For $k \geq 5$, R. Paturi, P. Pudlák and F. Zane [13] give the bound approaching $2^{(1-1/(2k))n}$ for large k. In this paper we improve all these bounds for deterministic algorithms.

Probabilistic Algorithm. Our algorithm is inspired by the currently fastest probabilistic 3-SAT algorithm of [15] which uses a well-known heuristic search method, cf. [2,9]. This algorithm is based on random walks (like Papadimitriou's algorithm [11] for 2-SAT). Given a formula F in k-CNF with n variables, the algorithm chooses exponentially many initial assignments at random and runs local search for each of them. Namely, if the assignment does not satisfy F, then the algorithm chooses any unsatisfied clause, chooses a literal from this clause at random, and flips its value. If a satisfying assignment is not found in $3n$ such steps, the algorithm starts local search from another random initial assignment.

Thus, the probabilistic algorithm of [15] includes two randomized components: (1) the choice of initial assignments, and (2) local search starting from these assignments.

Deterministic Algorithm. The deterministic k-SAT algorithm presented in this paper can be viewed as a derandomized version of the probabilistic algorithm above. The derandomization consists of two parts. To derandomize the choice of initial assignments, we cover the space of all possible 2^n assignments by balls of some Hamming radius r. We use coding theory to find a good covering for given r. In each ball, we run a deterministic version of local search to check whether there is a satisfying assignment inside the ball.

The optimal value of r can be chosen so that the overall running time is minimal. Taking $r = 1/(k+1)$, we obtain the running time $(2k/(k+1) + \varepsilon)^n$. We also show how to decrease this bound by using a more complicated version of local search. For 3-SAT, the modified algorithm gives the bound 1.481^n.

Notation. We consider propositional formulas in k-CNF ($k \geq 3$ is a constant). These formulas are conjunctions of clauses of size at most k. A *clause* is a disjunction of literals. A *literal* is a propositional variable or its negation. The size of a clause is the number of its literals. An *assignment* maps the propositional variables to the truth values $0, 1$, where 0 denotes $false$ and 1 denotes $true$. A *trivial* formula is the empty formula (which is always true) or a formula containing the empty clause (which is always false).

For an assignment a and a literal l, we write $a_{|l=1}$ to denote the assignment obtained from a by setting the value of l to 1 (more precisely, we set the value of the variable corresponding to l). We also write $F_{|l=1}$ to denote the formula obtained from F by assigning the value 1 to l, i.e., the clauses containing the

literal l itself are deleted from F, and the literal $\bar{l}$ is deleted from the other clauses.

We identify assignments with binary words. The *Hamming distance* between two assignments is the number of positions in which these two assignments differ. The *ball* of radius r around an assignment a is the set of all assignments whose Hamming distance to a is at most r.

A *code* of length n is a subset of $\{0,1\}^n$. The *covering radius* r of a code $\mathcal{C}$ is defined by

$$r = \max_{u \in \{0,1\}^n} \min_{v \in \mathcal{C}} d(u,v),$$

where $d(u,v)$ denotes the Hamming distance between u and v. The *normalized covering radius* ρ is defined by $\rho = r/n$.

Organization of the Paper. Section 2 describes the local search procedure. In Sect. 3 we specify the codes used in our algorithms. Section 4 contains the main algorithm and its analysis. We describe the technique yielding to the bound 1.481^n in Sect. 5. We discuss further developments in Sect. 6.

2 Local Search

Suppose we are given an initial assignment $a \in \{0,1\}^n$ where n is the number of variables. Consider the ball of radius r around a. Obviously, the volume of this ball is

$$V(n,r) = \sum_{i=0}^{r} \binom{n}{i}.$$

If the normalized radius $\rho = r/n$ satisfies $0 < \rho < 1/2$, the volume $V(n,r)$ can be estimated as follows [1, page 121] or [3, Lemma 2.4.4, page 33]:

$$\frac{1}{\sqrt{8n\rho(1-\rho)}} \cdot 2^{h(\rho)n} \;\leq\; V(n,r) \leq\; 2^{h(\rho)n} \tag{1}$$

where $h(\rho) = -\rho \log_2 \rho - (1-\rho)\log_2(1-\rho)$ is the binary entropy function. These bounds show that for a constant ρ, the volume $V(n,r)$ differs from $2^{h(\rho)n}$ at most by a polynomial factor.

The efficiency of our algorithm relies on the following important observation. Suppose we need to find a satisfying assignment inside the ball of radius r around a (if one exists). Then it is not necessary to search through all assignments inside this ball. The given formula F can be used to prune the search tree. The following easy lemma captures this observation.

Lemma 1. *Let F be a formula and a be an assignment such that F is false under a. Let C be an arbitrary clause in F that is false under a. Then F has a satisfying assignment belonging to the ball of radius r around a iff there is a literal l in C such that $F_{|l=1}$ has a satisfying assignment belonging to the ball of radius $r-1$ around a.*

Proof. For any clause C false under a, we have the following equivalence. The formula F has a satisfying assignment inside the ball of radius r around a iff there are a literal l in C and an assignment b such that l has the value 1 in b and the Hamming distance between b and $a_{|l=1}$ is at most $r-1$. The latter holds iff there is a literal l in C such that $F_{|l=1}$ has a satisfying assignment inside the ball of radius $r-1$ around a. □

The recursive procedure $Search(F, a, r)$ described below is the local search component of our algorithm. The procedure takes a formula F, an initial assignment a, and a radius r as input. It returns *true* if F has a satisfying assignment inside the ball of radius r around a. Otherwise, the procedure returns *false*.

Procedure $Search(F, a, r)$

1. If all clauses of F are true under a then return *true*.
2. If $r \leq 0$ then return *false*.
3. If F contains the empty clause then return *false*.
4. Pick (according to some deterministic rule) a clause C false under a. *Branch* on this clause C, i.e., for each literal l in C do the following: If $Search(F_{|l=1}, a, r-1)$ returns *false* then return *true*, otherwise return *false*.

The correctness of the procedure easily follows by induction on r with the invariant: $Search(F, a, r)$ outputs *true* iff F has a satisfying assignment inside the ball of radius r around a. For the induction step the lemma above is used.

The recursion depth of $Search(F, a, r)$ is at most r. If the input formula F is in k-CNF, the number of recursive calls generated at Step 4 is bounded by k. Therefore the recursion tree has at most k^r leaves and the complexity of the procedure is bounded by k^r.

Observe here that k^r can be much smaller than the volume of the ball of radius r around a. For example, for $r = n/2$ and $k = 3$ we obtain $V(n, r) \geq 2^{n-1}$ whereas $3^r < 1.733^n$. This gives us a very simple deterministic SAT algorithm: Given an input formula F with n variables, run $Search(F, a_0, n/2)$ and $Search(F, a_1, n/2)$, where a_0 and a_1 are n-bit assignments $a_0 = (0, 0, \ldots, 0)$ and $a_1 = (1, 1, \ldots, 1)$. Since any assignment has the Hamming distance $\leq n/2$ to either a_0 or a_1, the algorithm is correct. Its running time is bounded by 1.733^n.

The set $\{a_0, a_1\}$ of assignments in the example above is nothing else than a very special covering code. In the next section we show how this example can be generalized and improved.

3 Good Coverings

By a *covering* of radius r for $\{0,1\}^n$ we mean a code $\mathcal{C}$ of length n such that any word in $\{0,1\}^n$ belongs to at least one ball of radius r around a code word of $\mathcal{C}$.

For any covering $\mathcal{C}$ of radius r, we have $|\mathcal{C}| \cdot V(n, r) \geq 2^n$, where $V(n, r)$ is the volume of a ball of radius r. Using the upper bound on $V(n, r)$ in (1), we obtain

$$|\mathcal{C}| \geq \frac{2^n}{V(n, r)} \geq \frac{2^n}{2^{h(\rho)n}} = 2^{(1-h(\rho))n}$$

where $\rho = r/n$ is the normalized covering radius. Here the second inequality holds when $0 < \rho < 1/2$. This lower bound on $|\mathcal{C}|$ is known as the *sphere covering bound*. The following lemma from coding theory implies the existence of coverings whose cardinalities "almost" achieve the sphere covering bound.

Lemma 2 ([3, Theorem 12.1.2, page 320]). *For any n and r, there exists a code $\mathcal{C}$ of length n and covering radius r with*

$$|\mathcal{C}| \leq \lceil n2^n \ln 2/V(n,r) \rceil. \tag{2}$$

Corollary 1. *Let $\delta > 0$ and $0 < \rho < 1/2$. There exist an integer $\nu = \nu(\delta)$ and a code Γ of length ν with the following properties:*

1. *The code Γ has the covering radius $r \leq \rho\nu$.*
2. *The code Γ achieves the sphere covering bound up to the "$+\delta\nu$" in the exponent, namely: $|\Gamma| \leq 2^{(1-h(\rho)+\delta)\nu}$.*

Proof. We obtain the required bound on the cardinality by combining (2) and the lower bound on the volume in (1). □

The lemma and corollary above do not provide us with an efficient construction of good coverings. To construct them efficiently, we use the concept of the direct sum of two codes. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two codes of lengths n_1, n_2 and covering radii r_1, r_2 respectively. The *direct sum* of $\mathcal{C}_1$ and $\mathcal{C}_2$ is the code $\mathcal{C}$ of length n_1+n_2 that consists of all $|\mathcal{C}_1| \cdot |\mathcal{C}_2|$ possible concatenations w_1w_2, where $w_1 \in \mathcal{C}_1$ and $w_2 \in \mathcal{C}_2$. Obviously, the covering radius of $\mathcal{C}$ is $r_1 + r_2$.

To generate a good covering $\mathcal{C}$ for $\{0,1\}^n$, we fix $\varepsilon > \delta > 0$ and $0 < \rho < 1/2$. Let Γ be a code of length ν satisfying Corollary 1. The following algorithm takes a (large) length n as input and uses Γ to generate the code $\mathcal{C}$ of length n:

Algorithm for Generating Good Coverings

1. Find the smallest integer q such that $n \leq q\nu$.
2. Generate the direct sum Γ^q, i.e., generate words $w_1w_2 \ldots w_q$ one by one, where w_i ranges over Γ. If n is not divisible by ν, we restrict the code words of Γ^q to the first n positions.

The resulting code is the required covering $\mathcal{C}$. The algorithm runs in time polynomial in $|\mathcal{C}|$. If n (and q) is sufficiently large, its covering radius is at most $q\nu\rho \leq \lfloor n(\rho+\varepsilon) \rfloor$ and the cardinality $|\mathcal{C}|$ is bounded by:

$$2^{(1-h(\rho)+\delta)q\nu} \leq 2^{(1-h(\rho)+\varepsilon)n}.$$

Note that the sphere covering bound is "almost" (up to the "$+\varepsilon n$" in the covering radius and in the exponent) achieved. Note also that the code Γ provided by Corollary 1 is "hard wired" into the algorithm.

4 Main Algorithm and Its Analysis

Given fixed $\varepsilon > 0$ and $0 < \rho < 1/2$, our satisfiability algorithm takes as input a formula F with n variables and works as follows.

Main Algorithm

1. Generate a covering $\mathcal{C}$ for $\{0,1\}^n$ by using the algorithm for generating good coverings in Sect. 3.
2. Calculate $r = \lfloor n(\rho + \varepsilon) \rfloor$.
3. For each code word a in $\mathcal{C}$ run the procedure *Search*(F, a, r). Return *true* if at least one procedure call returns *true*. Otherwise return *false*.

If inputs are formulas in k-CNF, the complexity of the algorithm is bounded by $|\mathcal{C}| \cdot k^r$. Since $|\mathcal{C}| \leq 2^{(1-h(\rho))n}$ and $k^r \leq k^{n\rho}$ up to a factor of $c^{\varepsilon n}$, the complexity is bounded by

$$2^{(1-h(\rho))n} \cdot k^{n\rho} \;=\; \left(\frac{2k^\rho}{2^{h(\rho)}}\right)^n \;=\; (2\rho^\rho(1-\rho)^{1-\rho}k^\rho)^n.$$

The choice $\rho = 1/(k+1)$ minimizes (calculus required) the base value of this exponential function. Substituting $1/(k+1)$ for ρ, we have

$$2 \cdot \left(\frac{1}{k+1}\right)^{1/(k+1)} \cdot \left(1 - \frac{1}{k+1}\right)^{1-1/(k+1)} \cdot k^{1/(k+1)} \;=\; \frac{2k}{k+1}.$$

Thus, we obtain the following theorem.

Theorem 1. *For every $\varepsilon > 0$ there is a deterministic algorithm for deciding satisfiability of propositional formulas in k-CNF whose running time is*

$$\left(\frac{2k}{k+1} + \varepsilon\right)^n$$

where n is the number of variables in the input formula.

5 Improved Local Search

The local search procedure *Search*(F, a, r) from Sect. 2 picks *any* false clause for branching. The complexity of this procedure is at most k^r. Choosing a clause for branching more carefully, we can improve this bound and, thereby, the overall running time of our main algorithm. In this section we show how to improve *Search*(F, a, r) for formulas in 3-CNF so that its complexity is bounded by 2.848^r instead of 3^r. We then obtain the bound 1.481^n for 3-SAT.

Let F be formula and a an assignment. Let C be a clause in F. We classify C according to the number of its literals true under a. Namely, we call C a

$$(\underbrace{1,\ldots,1}_{p},\underbrace{0,\ldots,0}_{m})\text{-}\mathit{clause}$$

if C consists of exactly p literals true under a, and exactly m literals false under a. For example, a $(1,1,0)$-clause consists of two true and one false literals.

We say that F is *i-false* under a ($i \geq 0$) iff F has no satisfying assignment within Hamming distance i from a. When i is fixed, we can test in polynomial time whether F is i-false under a, where F and a are given as input.

The following observation follows from the fact that a (1)-clause $\bar{l}_i$ of F becomes the empty clause in $F_{|l_i=1}$.

Lemma 3. *Let F be a formula false under a. If $l_1 \vee l_2 \vee l_3$ is a $(0,0,0)$-clause and $\bar{l}_i$ is a (1)-clause of F then we have the following equivalence: F is satisfied within Hamming distance $\leq r$ from a iff there is a $j \neq i$ such that $F_{|l_j=1}$ is satisfied within Hamming distance $\leq r-1$ from a.*

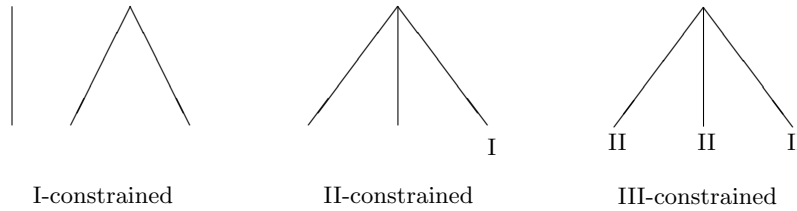


Fig. 1. Branching I, II, III-constrained formulas.

Let F be a non-trivial formula in 3-CNF. The following definitions describe types of formulas used in the new version of the procedure *Search*. Figure 1 illustrates these types.

1. Let F be false under a. We say that F is *I-constrained* with respect to a iff one of the following holds:
   - F has a (0)- or $(0,0)$-clause with respect to a.
   - F has a $(0,0,0)$-clause containing a literal l such that $\bar{l}$ is a (1)-clause of F.
2. Let F be 1-false under a. We say that F is *II-constrained* with respect to a iff one of the following holds:
   - F is I-constrained with respect to a.
   - F has a $(0,0,0)$-clause containing a literal l such that the formula $F_{|l=1}$ is I-constrained.
3. Let F be 2-false under a. We say that F is *III-constrained* with respect to a iff one of the following holds:
   - F is I-constrained or II-constrained with respect to a.
   - F has a $(0,0,0)$-clause such that for *each* literal l in this clause, $F_{l=1}$ is II-constrained.

The next lemma and its corollary show that there is no need to make recursive calls for not III-constrained formulas, because such formulas turn out to be satisfiable.

Lemma 4. *Let F be non-trivial and 3-false under a. Let l be a literal false under a. Let $F_{|l=1}$ still be non-trivial (it is 2-false under a by assumption). Then we have the implication: $F_{|l=1}$ is III-constrained with respect to a then $F_{|l=1}$ is II-constrained or F itself is III-constrained.*

Proof. First, a preparatory remark: If F is non-trivial, 1-false under a and not I-constrained then $F_{|l=1}$ is non-trivial for any literal l belonging to a clause of F false under a. This is because the empty clause could only be generated if F contained the clause $\bar{l}$, which is not the case as F is not I-constrained under a.

We show that if $F_{|l=1}$ is III-constrained with respect to a then $F_{|l=1}$ is II-constrained or F is III-constrained.

By assumption $F_{|l=1}$ is non-trivial and 2-false under a. If $F_{|l=1}$ is I- or II-constrained, the lemma holds. The remaining case to consider is that $F_{|l=1}$ has a $(0,0,0)$-clause $l_1 \vee l_2 \vee l_3$ such that each $F_{|l=1,l_i=1}$ is II-constrained. If an $F_{|l=1,l_i=1}$ is I-constrained then $F_{|l=1}$ is II-constrained and the lemma holds. We have to consider the situation that in each $F_{|l=1,l_i=1}$ we have a $(0,0,0)$-clause $k_i \vee k_{i,1} \vee k_{i,2}$ such that (without loss of generality) setting $k_i = 1$ shows that $F_{|l=1,l_i=1}$ is II-constrained. This means that $G_i = F_{|l=1,l_i=1,k_i=1}$ is I-constrained for $i = 1,2,3$. As F is 3-false under a, we obtain that G_i is false under a.

It follows from the remark in the beginning of our proof that G_i is non-trivial.

The final idea is as follows. A clause that causes G_i to be I-constrained is present in F or generated in 1 or 2 (but not 3 because of F is in 3-CNF) of the steps $l = 1, l_i = 1, k_i = 1$. If the clause is present in F we are done. If the step $l = 1$ is among the steps making at least one G_i I-constrained, $F_{|l=1}$ is already II-constrained. If for no G_i the step $l = 1$ is necessary to make G_i I-constrained, the clause $l_1 \vee l_2 \vee l_3$ and other clauses already present in F cause each G_i to be I-constrained. This means that F itself is III-constrained. The missing details of this argument follow below.

We consider several cases depending on the types of clauses that make G_i I-constrained:

Case of $(0,0)$-clause. If G_i has a $(0,0)$-clause $h_1 \vee h_2$ then $h_1 \vee h_2$ is generated when setting $k_i = 1$. This is because we assume that $F_{|l=1,l_i=1}$ is not I-constrained. Therefore $F_{|l=1,l_i=1}$ has a $(1,0,0)$-clause $\bar{k_i} \vee h_1 \vee h_2$ and the $(0,0,0)$-clause $k_i \vee k_{i,1} \vee k_{i,2}$ where $h_1, h_2 \neq k_i$. As F is in 3-CNF these clauses are also present in F and we obtain that F is already II-constrained.

Case of 0-clause. If G_i has a 0-clause h_i then h_i is generated when setting $k_i = 1$. Again this is because we assume that $F_{|l=1,l_i=1}$ is not I-constrained. Therefore $F_{|l=1,l_i=1}$ has a $(1,0)$-clause $\bar{k_i} \vee h_i$ and the $(0,0,0)$-clause $k_i \vee k_{i,1} \vee k_{i,2}$ where $h_i \neq k_i$. The clause $k_i \vee k_{i,1} \vee k_{i,2}$ is present in F and $F_{l=1}$. If $\bar{k_i} \vee h_i$ is also present in $F_{|l=1}$ then $F_{|l=1}$ is II-constrained and we are done.

We have to assume that $\bar{k_i} \vee h_i$ is not present in $F_{|l=1}$. Then $F_{|l=1}$ has the (0,0,0)-clauses

$$l_1 \vee l_2 \vee l_3$$
$$k_i \vee k_{i,1} \vee k_{i,2}$$

and the (1,1,0)-clause $\bar{l_i} \vee \bar{k_i} \vee h_i$. Since F is in 3-CNF, these clauses also belong to F.

Case of $(0,0,0)$-clause and 1-clause. Let $h'_i \vee h''_i \vee h'''_i$ be a $(0,0,0)$-clause with respect to a in G_i. Let $\bar{h'_j}$ be a 1-clause in G_i (without loss of generality). If $F_{|l=1,l_i=1}$ has the 1-clause $\bar{h'_i}$ then $F_{|l=1}$ is II-constrained and we are done.

So we assume that $F_{|l=1,l_i=1}$ does not have the clause $\bar{h'_i}$. Then $F_{|l=1,l_i=1}$ must have the clause $\bar{k_i} \vee \bar{h'_i}$. If this clause is also present in $F_{|l=1}$ then $F_{|l=1}$ is II-constrained and we are done.

We still need to assume that the clause $\bar{k_i} \vee \bar{h'_i}$ is not in $F_{|l=1}$. Then $F_{|l=1}$ has the (0,0,0)-clauses

$$l_1 \vee l_2 \vee l_3$$
$$k_i \vee k_{i,1} \vee k_{i,2}$$
$$h'_i \vee h''_i \vee h'''_i$$

and the (1,1,1)-clause $\bar{l_i} \vee \bar{k_i} \vee \bar{h'_i}$. Since F is in 3-CNF, these clauses are also present in F.

If we cannot conclude that F or $F_{|l=1}$ is II-constrained by now, we obtain that for each $i = 1, 2, 3$, one of the two remaining cases above applies. Then the $(0,0,0)$-clauses $l_1 \vee l_2 \vee l_3$, $k_i \vee k_{i,2} \vee k_{i,3}$ for $i = 1, 2, 3$, and the other clauses as specified above are present in F. These clauses show that F is III-constrained because when we branch on the clause $l_1 \vee l_2 \vee l_3$ instead of considering $F_{|l=1}$, we can see that each $F_{|l_i=1}$ is II-constrained. □

Corollary 2. *Let F be non-trivial and 2-false with respect to a. If F is not III-constrained with respect to a then F is satisfiable.*

Proof. If F is not 3-false under a then F is satisfiable and the corollary is proved. So assume F is 3-false and has clauses false under a. The only such clauses are $(0,0,0)$-clauses because F is not III-constrained and, therefore, not I-constrained with respect to a. Let $l_1 \vee l_2 \vee l_3$ be a $(0,0,0)$-clause with respect to a in F. Then each $F_{|l_i=1}$ is non-trivial and 2-false (cf. the remark in the beginning of the proof of Lemma 4). If each $F_{|l_i=1}$ is III-constrained, we obtain from Lemma 4 that each $F_{|l_i=1}$ is II-constrained or F itself is III-constrained. In any case F is III-constrained.

However F is not III-constrained by assumption. Therefore we obtain that at least one $F_{l_i=1}$ is non-trivial, 2-false under a, and not III-constrained. Moreover, $F_{l_i=1}$ has strictly less false clauses than F. Induction on the number of false clauses of F shows that finally we arrive at a formula that is not any more 3-false under a and hence satisfiable. □

Like the initial version of the procedure $Search(F, a, r)$ defined in Sect. 2, the new version takes as input a formula F in 3-CNF with n variables, an initial assignment a, and a radius r. It returns *true* if F has a satisfying assignment inside the ball of radius r around a. If F is unsatisfiable, the procedure returns *false*.

Procedure $Search(F, a, r)$

1. If F is not 2-false under a then return *true*.
2. If $r \leq 0$ then return *false*.
3. If F contains the empty clause then return *false*.
4. If F is not III-constrained with respect to a then return *true*.
5. If F is I-constrained with respect to a then branch on a false clause that certifies that F is I-constrained.
6. If F is II-constrained with respect to a then branch on a false clause that certifies that F is II-constrained.
7. Branch on a false clause that certifies that F is III-constrained.

Here the notion of branching is modified as follows. If we branch on a $(0,0,0)$-clause $l_1 \vee l_2 \vee l_3$ such that F has the 1-clause $\bar{l}_i$ then we do not run $Search(F_{l_i=1}, a, r-1)$.

The correctness of the procedure follows from Lemma 1 and 2 by induction on r. To estimate the number of leaves of the recursion tree, we use the function H defined by recursion as follows: $H(0) = 1$, $H(1) = 3$, $H(2) = 9$, and for $r \geq 3$

$$H(r) = 6 \cdot (H(r-2) + H(r-3)).$$

Lemma 5. *Let $L(F, a, r)$ be the number of leaves of the recursion tree of the procedure $Search(F, a, r)$. Then we have $L(F, a, r) \leq H(r)$ for all r.*

Proof. We first show that $2 \cdot H(r-1) \leq H(r)$ for all $r \geq 1$. This holds for $r = 1$, $r = 2$ and $r = 3$. For $r > 3$ the inequality is proved by easy induction. Second we show that $2 \cdot (H(r-1) + H(r-2)) \leq H(r)$ for $r \geq 2$. This can be calculated directly for $r = 2, 3, 4$. For $r > 4$ we use induction:

$$\begin{aligned}
&2 \cdot H(r-1) + 2 \cdot H(r-2) \\
&= 12 \cdot H(r-3) + 12 \cdot H(r-4) + 12 \cdot H(r-4) + 12 \cdot H(r-5) \\
&\leq H(r) \quad \text{(induction hypothesis)}
\end{aligned}$$

The claim of the lemma now holds for $r = 0, 1, 2$. For induction on r we proceed as follows. If F is not 2-false under a, we have one leaf in the recursion tree of $Search(F, a, r)$ and the claim holds. The same applies when F has the empty clause. If F is not III-constrained with respect to a, we again have only one leaf and the claim holds.

Otherwise F is I-, II-, or III-constrained. The claim follows by induction, applying the inequalities above and the definition of H. □

To obtain an explicit bound on the size of the recursion tree, we solve the recurrence for H. If α satisfies the equation $\alpha^3 = 6 \cdot \alpha + 6$, we assume $H(m) = \alpha^m$ for $m \leq r$ and derive $H(r+1) = \alpha^{r+1}$. The initial conditions are taken care of when we bound $H(r) \leq 9 \cdot \alpha^r$ for all r where α is the unique (some calculus required) positive solution of the cubic equation above. One can calculate that

$\alpha = \sqrt[3]{4} + \sqrt[3]{2}$ which is between 2.847 and 2.848. Hence the induction base holds. For the induction step observe that

$$\begin{aligned} H(r+1) &\leq 6 \cdot 9 \cdot \alpha^{r-1} + 6 \cdot 9 \cdot \alpha^{r-2} \quad \text{(induction hypothesis)} \\ &= 9 \cdot \alpha^{r+1} \end{aligned}$$

Choosing $\rho = 0.26$, we obtain a satisfiability algorithm running in time exponential in n where the base of the exponent is $2^{\varepsilon n}$ times

$$\left(2.848^{0.26} \cdot 2^{1-h(0.26)}\right)^n .$$

Therefore, the running time of our 3-SAT algorithm is bounded by 1.481^n.

6 Conclusion

In [15] the fourth author has shown that the idea of local search yields probabilistic algorithms whose running time is the best known for probabilistic 3-SAT algorithms. Here we show that local search can be also used to obtain fast deterministic algorithms for k-SAT. Similarly to [15], it is possible to extend our approach to the more general class of constraint satisfaction problems. In contrast to other deterministic k-SAT algorithms, our basic algorithm presented in Sect. 4 is technically very simple and has a better running time.

The improvement for 3-SAT given in Sect. 5 can be generalized to k-SAT. It is an open problem whether this method can be used to improve the complexity of the probabilistic algorithm from [15] (either for 3-SAT or for k-SAT).

About This Paper

This paper resulted from merging two independent papers submitted simultaneously to two different conferences. Also, we recently learned that R. Kannan, J. Kleinberg, C. H. Papadimitriou and P. Raghavan independently obtained results similar to our results in Sect. 2–4.

U. Schöning wants to thank V. Arvind, S. Baumer, M. Bossert, J. Köbler, and P. Pudlák for valuable remarks and discussions. A. Goerdt thanks Oliver Kullmann for answering some questions. E. Dantsin and E. A. Hirsch are very grateful to Simon Litsyn who helped them to choose appropriate covering codes. They also thank Dima Grigoriev, Yuri Matiyasevich, Andrei Voronkov, and Maxim Vsemirnov for helpful discussions.

References

1. R. B. Ash. *Information Theory.* Dover, 1965.
2. L. Bolc and J. Cytowski. *Search Methods for Artificial Intelligence.* Academic Press, 1992.

3. G. Cohen, I. Honkala, S. Litsyn, and A. Lobstein. *Covering Codes*, volume 54 of *Mathematical Library*. Elsevier, 1997.
4. E. Dantsin. Two propositional proof systems based on the splitting method (in Russian). *Zapiski Nauchnykh Seminarov LOMI*, 105:24–44, 1981. English translation: *Journal of Soviet Mathematics*, 22(3):1293–1305, 1983.
5. E. A. Hirsch. Two new upper bounds for SAT. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA'98*, pages 521–530, 1998. A journal version is to appear in *Journal of Automated Reasoning*, 2000.
6. O. Kullmann. Worst-case analysis, 3-SAT decision and lower bounds: approaches for improved SAT algorithms. In D. Du, J. Gu, and P. M. Pardalos, editors, *Satisfiability Problem: Theory and Applications (DIMACS Workshop March 11-13, 1996)*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 261–313. AMS, 1997.
7. O. Kullmann. New methods for 3-SAT decision and worst-case analysis. *Theoretical Computer Science*, 223(1-2):1–72, 1999.
8. O. Kullmann and H. Luckhardt. Deciding propositional tautologies: Algorithms and their complexity. Preprint, 82 pages, http://www.cs.toronto.edu/~kullmann. A journal version is submitted to Information and Computation, January 1997.
9. S. Minton, M. D. Johnston, A. B. Philips, and P. Lair. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.
10. B. Monien and E. Speckenmeyer. Solving satisfiability in less then 2^n steps. *Discrete Applied Mathematics*, 10:287–295, 1985.
11. C. H. Papadimitriou. On selecting a satisfying truth assignment. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science, FOCS'91*, pages 163–169, 1991.
12. R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, FOCS'98*, pages 628–637, 1998.
13. R. Paturi, P. Pudlák, and F. Zane. Satisfiability coding lemma. In *Proceedings of the 38th Annual IEEE Symposium on Foundations of Computer Science, FOCS'97*, pages 566–574, 1997.
14. I. Schiermeyer. Pure literal look ahead: An $O(1,497^n)$ 3-satisfiability algorithm. Workshop on the Satisfiability Problem, Siena, April 29 – May 3, 1996. Technical Report 96-230, University Köln, 1996.
15. U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, FOCS'99*, pages 410–414, 1999.
16. W. Zhang. Number of models and satisfiability of sets of clauses. *Theoretical Computer Science*, 155:277–288, 1996.

Closest Vectors, Successive Minima, and Dual HKZ-Bases of Lattices

Johannes Blömer*

Department of Mathematics and Computer Science, University of Paderborn,
D-33095 Paderborn, Germany
bloemer@uni-paderborn.de.

Abstract. In this paper we introduce a new technique to solve lattice problems. The technique is based on dual HKZ-bases. Using this technique we show how to solve the closest vector problem in lattices with rank n in time $n! \cdot s^{O(1)}$, where s is the input size of the problem. This is an exponential improvement over an algorithm due to Kannan and Helfrich [16,15]. Based on the new technique we also show how to compute the successive minima of a lattice in time $n! \cdot 3^n \cdot s^{O(1)}$, where n is the rank of the lattice and s is the input size of the lattice. The problem of computing the successive minima plays an important role in Ajtai's worst-case to average-case reduction for lattice problems. Our results reveal a close connection between the closest vector problem and the problem of computing the successive minima of a lattice.

1 Introduction

In this paper we study two classical problems from the geometry of numbers, the *closest vector problem* (Cvp) and the *shortest linearly independent vectors problem* (Sivp). In the closest vector problem we are given a lattice L and some vector $\mathbf{t}$ in the $\mathbb{R}$-vector space span(L) spanned by the vectors in L. We are asked to find a vector $\mathbf{u} \in \mathsf{L}$, whose Euclidean distance to $\mathbf{t}$ is as small as possible. To define the second problem, first we need to define the successive minima $\lambda_k(\mathsf{L})$ of a lattice of L. We call the dimension of span(L) the rank of L. Let k be an integer less than or equal to the rank of L. The *k-th successive minimum of* L is the smallest real number r such that L contains k linearly independent vectors of Euclidean length at most r. In the shortest linearly independent vectors problems we are given a lattice L with rank n. We are asked to find n linearly independent vectors $\mathbf{v}_1, \ldots, \mathbf{v}_n$ such that the length of $\mathbf{v}_k$ is $\lambda_k(\mathsf{L})$. It is a well-known fact, that vectors with this property always exist (see [9]).

Next to the shortest vector problem, Cvp is the most intensively studied problem in the algorithmic geometry of numbers. Cvp is an NP-hard problem (see for example [16]). Cvp is also hard to approximate (see [3,4,13]). The best result in this direction is due to [13], who show that Cvp can not be approximated within a factor of $n^{1/\log\log(n)}$, where n is the rank of the lattice. On the

* Work done while at Institut für theoretische Informatik, ETH Zürich, Switzerland

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 248–259, 2000.
© Springer-Verlag Berlin Heidelberg 2000

positive side, Babai [6] showed that in polynomial time the distance of a vector $\mathbf{t} \in \mathrm{span}(\mathsf{L})$ to its closest vectors in L can be approximated within a factor of $(3/\sqrt{2})^n$. Babai's algorithm is based on the LLL-algorithm. The best algorithm to determine the exact distance from $\mathbf{t} \in \mathrm{span}(\mathsf{L})$ to L is due to Kannan [16], with improvements partly due to Helfrich [15]. Kannan's algorithm achieves a running time of $n^n \cdot s^{O(1)}$, where n is the rank of L and s is the representation size of $\mathbf{t}$ and L. Kannan's algorithm is based on so-called HKZ-bases for lattices. An HKZ-basis for L is a basis for L consisting of "short" vectors (a precise definition is given in Section 2).

The first result of this paper is an improvement of Kannan's algorithm. We describe an algorithm that computes a vector in L closest to $\mathbf{t}$ in time $n! \cdot s^{O(1)}$. Since $n^n/n! \approx \mathrm{e}^n$, this is an exponential improvement over Kannan's result. The new algorithm is simple and, unlike previous algorithms for lattice problems, it uses *dual HKZ-bases*. A dual HKZ-basis is a basis, whose dual is a HKZ-basis of the dual lattice L^* (see Section 2 for exact definitions). The algorithm is based on the following two facts.

1. Assume $\mathbf{t} = \sum_{j=1}^{n} q_j \mathbf{b}_j, q_j \in \mathbb{Q}$, where $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ is a dual HKZ-basis for L.

   If $\mathbf{u} = \sum_{j=1}^{n} c_j \mathbf{b}_j, c_j \in \mathbb{Z}$, is a vector in L closest to $\mathbf{t}$, then
   $$|q_n - c_n| \leq n/2. \tag{1}$$
2. Assume $c \in \mathbb{Z}$ and $\mathbf{v}$ is a vector in the lattice L' spanned by the vectors $[\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}]$ such that $c \cdot \mathbf{b}_n + \mathbf{v}$ is a vector in L closest to $\mathbf{t}$, then $\mathbf{v}$ is a vector in L' closest to the orthogonal projection of $\mathbf{t} - c \cdot \mathbf{b}_n$ onto $\mathrm{span}(\mathsf{L}')$.

The second fact is straightforward. The first one follows from so-called transference bounds. Transference bounds relate fundamental constants of a lattice L and its dual L^* (see [7,18]). From the above mentioned facts one gets the following algorithm for CVP. For all integers c with $|q_n - c| \leq n/2$, recursively compute a vector $\mathbf{v}_c$ in L' closest to the orthogonal projection of $\mathbf{t} - c \cdot \mathbf{b}_n$ onto L'. Among the vectors $\mathbf{v}_c + c \cdot \mathbf{b}_n$ determine one closest to $\mathbf{t}$.

The new technique based on dual HKZ-bases also sheds light on the complexity of SIVP. Although the successive minima of a lattice are a classical notion in the geometry of numbers, until recently the complexity of SIVP has received little attention. This changed with Ajtai's discovery of the connection between the average-case complexity of the *shortest vector problem* (SVP) and the worst-case complexity of SIVP (see [1,2,12,10,11] among others). This connection also opened up the possibility of developing provably secure cryptographic primitives based on the single assumption $\mathsf{P} \neq \mathsf{NP}$ (see for example [5]). Hence, understanding the complexity of SIVP is an important issue in complexity theory as well as cryptography.

In [8] it was shown that SIVP is NP-hard. Moreover, it was shown, that $\lambda_n(\mathsf{L})$ is hard to approximate within a factor of $n^{1/2 \log\log(n)}$. Here n is the

rank of the lattice L. On the positive side, the LLL-algorithm approximates all successive minima of a lattice within a factor of $2^{n/2}$. The only previously published algorithm, we are aware of, that solves SIVP exactly is due to Pohst [19]. However, this algorithm predates the LLL-algorithm and no analysis for the algorithm was given. An algorithm of Helfrich [15] to compute a so-called Minkowski-basis of a lattice can be modified in order to solve SIVP. This modified algorithm solves SIVP in time $n^{n^2} \cdot s^{O(1)}$, where n is the rank of L and s its representation size.

In this paper, we observe that SIVP closely resembles CVP. In fact, the problem SIVP is a special case of the following generalization of CVP. As in CVP, we are given a lattice L and a vector $\mathbf{t} \in \mathrm{span}(\mathsf{L})$. Moreover, we are given an affine space $\mathrm{A} \subset \mathrm{span}(\mathsf{L})$. We are asked to find a vector $\mathbf{u}$ in $\mathsf{L} \backslash \mathrm{A}$ closest to $\mathbf{t}$. That is, we are looking for a closest vector to $\mathbf{t}$ in L avoiding the affine space A. We call this problem the *generalized closest vector problem* (GCVP). The connection between GCVP and SIVP is as follows. Suppose we already know linearly independent lattice vectors $\mathbf{v}_1, \ldots, \mathbf{v}_{k-1}$, where the length of $\mathbf{v}_i$ is $\lambda_i(\mathsf{L})$. It is well-known that there is always a lattice vector $\mathbf{v}_k$ linearly independent from $\mathbf{v}_i, i \leq k-1$, with length $\lambda_k(\mathsf{L})$. We can characterize $\mathbf{v}_k$ as follows. $\mathbf{v}_k$ is independent from $\mathbf{v}_i, i \leq k-1$, iff $\mathbf{v}_k \notin \mathrm{S} = \mathrm{span}(\mathbf{v}_1, \ldots, \mathbf{v}_{k-1})$. Moreover, it must be a vector in $\mathsf{L} \backslash \mathrm{S}$ closest to the origin $\mathbf{0}$. Hence, $\mathbf{v}_k$ can be described as a solution to an instance of GCVP. It might seem more natural to describe $\mathbf{v}_k$ as a *shortest vector* in $\mathsf{L} \backslash \mathrm{S}$. However, we will give a recursive algorithm for SIVP and GCVP that closely resembles the algorithm for CVP described above. In this recursive algorithm, even if we start with a linear subspace S and target vector $\mathbf{0}$, we need to solve instances of GCVP with arbitrary target vectors and arbitrary affine subspaces.

Our algorithm for SIVP and GCVP achieves a running time of $3^n \cdot n! \cdot s^{O(1)}$. This differs from the running time for CVP by a factor of 3^n. This difference is caused by a weaker version of equation (1) for GCVP. The running time we achieve is exponentially better than the running time one achieves by using the technique of Helfrich based on HKZ-bases. Moreover, the algorithm indicates that SIVP is at least as difficult as CVP.

Let us briefly summarize the main contributions of this paper.

1. We introduce a new technique for solving lattice problems. This technique is based on dual HKZ-bases.
2. Based on the technique, we give an improved algorithm for the closest vector problem.
3. We give a new algorithm for the shortest independent vectors problem.
4. We show a close connection between SIVP and GCVP. This in turn hints at a connection between SIVP and CVP.

The last point immediately leads to the main open question this paper raises. Is the problem SIVP polynomial-time reducible to CVP, or vice versa? More generally, is GCVP polynomial-time reducible to CVP? It would also be interesting to see whether other lattice problems can be solved using dual HKZ-bases. A good candidate is the computation of a Minkowski-basis of a lattice (see [15]).

The paper is organized as follows. In Section 2, we state the most important facts used in this paper. In Section 3, we describe the algorithm for the closest vector problem. The solution to the shortest independent vectors problem and the generalized closest vector problem is given in Section 4.

2 Basis Definitions and Facts

In this section we define several fundamental concepts and state important results from the geometry of numbers that will be used throughout this paper. Given a lattice L and a target vector $\mathbf{t} \in \operatorname{span}(\mathsf{L})$, by $\mu(\mathbf{t}, \mathsf{L})$ denote the Euclidean distance of a closest vector in L to $\mathbf{t}$.

Definition 1. *The number* $\mu(\mathsf{L}) := \max_{\mathbf{t} \in \operatorname{span}(\mathsf{L})} \mu(\mathbf{t}, \mathsf{L})$ *is called the covering radius of* L.

Given a lattice L and a basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of L, we denote the sublattice with basis $[\mathbf{b}_1, \ldots, \mathbf{b}_k]$ by L_k, $k = 1, \ldots, n$. Hence $\mathsf{L}_n = \mathsf{L}$. The *dual lattice* L^* of L is defined as

$$\{\mathbf{v} \in \operatorname{span}(\mathsf{L}) : \langle \mathbf{v}, \mathbf{w} \rangle \in \mathbb{Z} \text{ for all } \mathbf{w} \in \mathsf{L}\}.$$

For a basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of L there is a unique basis $[\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*]$ of L^* satisfying

$$\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle = \begin{cases} 1 & \text{if } i + j = n + 1, \\ 0 & \text{otherwise.} \end{cases}$$

This basis is called a *dual basis* to the basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$.

The following theorem is the most important ingredient in the analysis of our algorithms. Results of this type are usually referred to as *transference bounds*, since they relate fundamental constants of L to fundamental constants of L^*. The version we state here is due to Banaszczyk [7]. Weaker versions were proved by Lagarias, Lenstra, and Schnorr [18].

Theorem 1 (Transference bounds). *For every lattice* L *of rank* n *and its dual* L^* *the following inequalities hold*

1. $\lambda_i(\mathsf{L}) \cdot \lambda_{n-i+1}(\mathsf{L}^*) \leq n, \ i = 1, \ldots, n.$
2. $\mu(\mathsf{L}) \cdot \lambda_1(\mathsf{L}^*) \leq n/2.$

For $n \to \infty$, the right-hand sides in the theorem both can be replaced by $n/(2\pi)$. Using these bounds the running times of our algorithms can be improved. However, in order to keep the exposition simple, we will use the bounds given in Theorem 1, valid for any lattice.

For a lattice L in $\mathbb{R}^m$ with basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ and for $1 \leq i \leq n$ we define the lattices $\mathsf{L}^{(n-i+1)} := \pi_i(\mathsf{L})$, where $\pi_i : \mathbb{R}^m \to \operatorname{span}(L_{i-1})^\perp$ denotes the orthogonal projection onto the orthogonal complement of $\operatorname{span}(\mathsf{L}_{i-1})$. The lattice $\mathsf{L}^{(n-i+1)}$ has rank $n-i+1$ and a basis is given by $[\pi_i(\mathbf{b}_i), \ldots, \pi_i(\mathbf{b}_n)]$. For a vector $\mathbf{v} \in \mathsf{L}$ we denote $\pi_i(\mathbf{v})$ by $\mathbf{v}(i)$. Moreover, $\pi_i(\mathbf{b}_i) = \mathbf{b}_i^\dagger = \mathbf{b}_i(i)$ for basis vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$.

The vectors $[\mathbf{b}_1^\dagger, \dots, \mathbf{b}_n^\dagger]$ are the *Gram-Schmidt orthogonalization* of $[\mathbf{b}_1, \dots, \mathbf{b}_n]$. The numbers

$$\mu_{ij} := \frac{\langle \mathbf{b}_i, \mathbf{b}_j^\dagger \rangle}{\langle \mathbf{b}_i^\dagger, \mathbf{b}_j^\dagger \rangle}, 1 \le j < i \le n,$$

are called the Gram-Schmidt coefficients of $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ A basis $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ is called *weakly reduced* iff $|\mu_{ij}| \le 1/2$ for $1 \le j < i \le n$.

Definition 2. *A basis* $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ *of a lattice* L *is called an* HKZ-basis *(Hermite, Korkin, Zolotarev) iff*

(i) $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ *is weakly reduced.*
(ii) $\mathbf{b}_i^\dagger$ *is a shortest non-zero vector in* $\mathsf{L}^{(n-i+1)}, i = 1, \dots, n$.

A basis $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ *of a lattice* L *is called a* dual HKZ-basis*, iff the dual basis* $[\mathbf{b}_1^*, \dots, \mathbf{b}_n^*]$ *is an HKZ-basis of the dual lattice* L^*.

By (ii), the first vector $\mathbf{b}_1$ of an HKZ-basis $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ is a shortest vector of the lattice L. The following lemma will prove to be useful. It was used without proof in [18]. A proof for it will be contained in the full version of this paper.

Lemma 1. *If* $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ *is a dual HKZ-basis for* L*, then* $[\mathbf{b}_1, \dots, \mathbf{b}_k]$ *is a dual HKZ-basis for* L_k*,* $k = 1, \dots, n$.

Next we need to say something about representations and representation sizes of vectors, lattices, and affine subspaces. We always assume that a lattice $\mathsf{L} \subset \mathbb{Q}^m$ is given by a basis $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ of L. The representation size s of a lattice $\mathsf{L} \subset \mathbb{Q}^m$ with respect to the basis $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ is the maximum of m and the binary lengths of the numerators and denominators of the coordinates of the basis vectors $\mathbf{b}_j$. Given a vector $\mathbf{t} = \sum_{j=1}^{n} q_j \mathbf{b}_j, q_j \in \mathbb{Q}$, the representation size of $\mathbf{t}$ with respect to $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ is the maximum of m and of the binary lengths of the numerators and denominators of the coefficients q_j. In the sequel, if we speak of the representation size of a lattice L, or of a vector $\mathbf{t} \in \mathrm{span}(\mathsf{L})$, without referring to some specific basis, we implicitly assume that some basis $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ for L is given. Finally, an affine space A is represented as $\mathbf{u} + \mathrm{S}$ for some vector $\mathbf{u} \in \mathbb{Q}^m$ and some linear subspace S. The subspace S is represented by some basis $[\mathbf{w}_1, \dots, \mathbf{w}_k], k \le m$. The representation size of A with respect to $\mathbf{u}$ and S is the maximum of m and the binary lengths of the numerators and denominators of the coordinates of the vectors $\mathbf{u}, \mathbf{w}_j$.

The only algorithmic tool we need in this paper is an algorithm to compute an HKZ-basis for a lattice L. Recall that the first vector of an HKZ-basis of L is a shortest vector in L. Incorporating improvements by Helfrich, an algorithm by Kannan obtains the following running times (see [16,15]).

Theorem 2. *Given a lattice* $\mathsf{L} \in \mathbb{Q}^m$ *with* $\mathrm{rank}(\mathsf{L}) = n$ *and representation size* s*, then an HKZ-basis of* L *can be computed with* $n^{n/2} \cdot s^{O(1)}$ *arithmetic operations on integers of size* $s^{O(1)}$*. In particular, a shortest vector in* L *can be computed within the time bounds stated above.*

3 The Closest Vector Problem

In this section we present the algorithm solving the closest vector problem. In the following section we will present the algorithm for the generalized closest vector problem. The algorithms are almost identical. But the basic ideas are clearer in case of the (classical) closest vector problem. Also, the running time we achieve for the closest vector problem is better than for the generalized closest vector problem. First we state and prove the lemmas crucial to our approach as outlined in Section 1.

Lemma 2. *Let* L *be a lattice of rank* n *with dual HKZ-basis* $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ *and let* $\mathbf{t}$ *be a vector in* $\operatorname{span}(\mathsf{L})$, $\mathbf{t} = \sum_{j=1}^{n} q_j \mathbf{b}_j, q_j \in \mathbb{Q}$. *If* $\mathbf{u} = \sum_{j=1}^{n} c_j \mathbf{b}_j, c_j \in \mathbb{Z}$, *is a vector in* L *closest to* $\mathbf{t}$, *then*

$$|q_n - c_n| \leq n/2.$$

Proof. Let $[\mathbf{b}_1^*, \dots, \mathbf{b}_n^*]$ be the basis dual to $[\mathbf{b}_1, \dots, \mathbf{b}_n]$. By definition of a dual basis, $|q_n - c_n| = |\langle \mathbf{t} - \mathbf{u}, \mathbf{b}_1^* \rangle|$. Recall that the first vector in an HKZ-basis of a lattice is a shortest vector of the lattice. Hence, in the present case, $\|\mathbf{b}_1^*\| = \lambda_1(\mathsf{L}^*)$. Applying the Cauchy-Schwarz-inequality, we obtain

$$|q_n - c_n| = |\langle \mathbf{t} - \mathbf{u}, \mathbf{b}_1^* \rangle| \leq \|\mathbf{t} - \mathbf{u}\| \cdot \|\mathbf{b}_1^*\| \leq \mu(\mathsf{L}) \cdot \lambda_1(\mathsf{L}^*).$$

The transference bound (Theorem 1) proves the lemma. □

The next lemma is the key to the recursive step of our algorithm for the closest vector problem.

Lemma 3. *Let* L *be a lattice of rank* n *with dual HKZ-basis* $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ *and let* $\mathbf{t}$ *be a vector in* $\operatorname{span}(\mathsf{L})$. *Fix some integer* c *and denote by* $\mathbf{w}$ *the orthogonal projection of* $\mathbf{t} - c \cdot \mathbf{b}_n, c \in \mathbb{Z}$, *onto* $\operatorname{span}(\mathsf{L}_{n-1})$. *The vector* $\mathbf{v} + c \cdot \mathbf{b}_n, \mathbf{v} \in \mathsf{L}_{n-1}$, *is a closest vector to* $\mathbf{t}$ *of the form* $\mathbf{u} + c \cdot \mathbf{b}_n, \mathbf{u} \in \mathsf{L}_{n-1}$, *iff* $\mathbf{v}$ *is a vector in* L_{n-1} *closest to* $\mathbf{w}$.

Proof. By definition $\mathbf{w} = \mathbf{t} - c \cdot \mathbf{b}_n - \mathbf{t}(n) + c \cdot \mathbf{b}_n^\dagger$. Consider an arbitrary vector of the form $\mathbf{u} + c \cdot \mathbf{b}_n, \mathbf{u} \in \mathsf{L}_{n-1}$. Since $\mathbf{u}(n) = \mathbf{u}$, the squared distance from $\mathbf{u} + c \cdot \mathbf{b}_n$ to $\mathbf{t}$ is given by $\|\mathbf{t} - c \cdot \mathbf{b}_n - \mathbf{u}\|^2 = \|\mathbf{t}(n) - c \cdot \mathbf{b}_n^\dagger\|^2 + \|\mathbf{w} - \mathbf{u}\|^2$. The term $\|\mathbf{t}(n) - c \cdot \mathbf{b}_n^\dagger\|^2$ is independent of the choice of $\mathbf{u}$. Hence, for $\mathbf{v} \in \mathsf{L}_{n-1}$ we get

$$\|\mathbf{v} + c \cdot \mathbf{b}_n - \mathbf{t}\| = \min_{\mathbf{u} \in \mathsf{L}_{n-1}} \|\mathbf{u} + c \cdot \mathbf{b}_n - \mathbf{t}\| \Longleftrightarrow \|\mathbf{v} - \mathbf{w}\| = \min_{\mathbf{u} \in \mathsf{L}_{n-1}} \|\mathbf{u} - \mathbf{w}\|.$$

This proves the lemma. □

Now we are in a position to state the algorithm for the closest vector problem, prove its correctness, and analyze its running time. The input to the algorithm is a dual HKZ-basis $[\mathbf{b}_1, \dots, \mathbf{b}_n]$ of the lattice L and a target vector $\mathbf{t} = \sum_{j=1}^{n} q_j \mathbf{b}_j, q_j \in \mathbb{Q}$.

Algorithm Closest Vector

```
IF n = 1 THEN
    Compute the closest integer c to q_1 and return c · b_1.
ELSE
    Set C := ∅.
    FOR c = −n/2 TO n/2 DO
        Compute the orthogonal projection w of t − c · b_n onto span(L_{n−1}).
        Recursively call Algorithm Closest Vector with input basis
        [b_1, ... , b_{n−1}] and target vector w.
        Set C := C ∪ {v + c · b_n}, where v is the vector returned by the
        recursive call.
    END
    Compute and return an element in C closest to t.
END
```

Next we show the correctness of the algorithm.

Lemma 4. *Assume the input to Algorithm Closest Vector is a dual HKZ-basis* $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ *of the lattice* L *and a target vector* $\mathbf{t} = \sum_{j=1}^{n} q_j \mathbf{b}_j, q_j \in \mathbb{Q}$. *Then the algorithm computes a vector in* L *closest to* $\mathbf{t}$.

Proof. We show the correctness of the algorithm by induction on the rank n. For $n = 1$ the algorithm correctly computes closest vectors. So assume the algorithm correctly computes closest vectors for all lattices of rank $< n$. Now assume L is a lattice with rank n. By Lemma 1, $[\mathbf{b}_1, \ldots, \mathbf{b}_{n-1}]$ is a dual HKZ-basis of L_{n-1} and, by induction assumption, the recursive calls to the algorithm return closest vectors to the orthogonal projections of $\mathbf{t} - c \cdot \mathbf{b}_n, c \in \mathbb{Z}, |c| \leq n/2$. By Lemma 2 and Lemma 3, a vector in L closest to $\mathbf{t}$ can be written as $\mathbf{u} + c \cdot \mathbf{b}_n$ with $|c| \leq n/2$ and $\mathbf{u}$ an arbitrary vector in L_{n-1} closest to the orthogonal projection of $\mathbf{t} - c \cdot \mathbf{b}_n$ onto $\mathrm{span}(\mathsf{L}_{n-1})$. Hence the last line in the algorithm correctly computes a vector in L closest to $\mathbf{t}$. □

The running time of Algorithm Closest Vector is analyzed as follows.

Lemma 5. *Assume the lattice* $\mathsf{L} \subset \mathbb{R}^m$ *and the target vector* $\mathbf{t}$ *have representation size* s *with respect to the dual HKZ-basis* $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$. *Then Algorithm Closest Vector uses at most* $n! \cdot m^{O(1)}$ *arithmetic operations on integers of size* $s^{O(1)}$.

Proof. By $T(n, m)$ denote the maximal number of arithmetic operations Algorithm Closest Vector uses for lattices $\mathsf{L} \subset \mathbb{R}^m$ with rank n and target vectors $\mathbf{t} \in \mathrm{span}(\mathsf{L})$. Since projections can be computed with $m^{O(1)}$ arithmetic operations and the last step in Algorithm Closest Vector also requires $m^{O(1)}$ arithmetic operations, for $T(n, m)$ we obtain the following recurrence relation

$$T(n, m) = n \cdot T(n-1, m) + m^{O(1)}.$$

For lattices $\mathsf{L} \subset \mathbb{R}^m$ with rank 1 the algorithm uses $m^{O(1)}$ arithmetic operations. Hence, the recurrence for $T(n, m)$ solves to $n! \cdot m^{O(1)}$, proving our claim on the number of arithmetic operations.

The analysis of the size of the integers involved is straightforward and omitted in this extended abstract. □

For Algorithm Closest Vector we assume that we are already given a dual HKZ-basis of L and that the target vector $\mathbf{t}$ is represented as a linear combination of the vectors in the dual HKZ-basis. However, combining Algorithm Closest Vector with the Kannan/Helfrich algorithm to compute HKZ-bases we obtain the following theorem.

Theorem 3. *Given a lattice* L *with rank* n *and a vector* $\mathbf{t} \in \operatorname{span}(\mathsf{L})$ *both with representation size* s*. A closest vector to* $\mathbf{t}$ *in the lattice* L *can be computed with* $n! \cdot m^{O(1)} + n^{n/2} \cdot s^{O(1)}$ *arithmetic operations on integers of size* $s^{O(1)}$*.*

Proof. Since L has representation size s, the dual lattice L^* has representation size $s^{O(1)}$. By Theorem 2, an HKZ-basis $[\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*]$ for L^* can be computed with $n^{n/2} s^{O(1)}$ arithmetic operations on integers of size $s^{O(1)}$. From this basis within the bounds stated in the theorem we compute a dual HKZ-basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of L. Also the representation of $\mathbf{t}$ with respect to the basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ can be computed within the time bounds stated in the theorem. In particular, with respect to the new basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ the lattice L and the vector $\mathbf{t}$ have representation size $s^{O(1)}$. Next we apply Algorithm Closest Vector to the dual HKZ-basis computed previously and target vector $\mathbf{t}$ to obtain a vector in L closest to $\mathbf{t}$. The theorem follows from Lemma 5. □

Note that once an HKZ-basis for L^* has been computed, the number of arithmetic operations used by the algorithm is independent of s. This is a feature our algorithms shares with Kannan's algorithm for the closest vector problem. Kannan's algorithm needs $n^n \cdot m^{O(1)} + n^{n/2} \cdot s^{O(1)}$ arithmetic operations on integers of size $s^{O(1)}$. Hence our algorithms uses a number of arithmetic operations that is by a factor of roughly $n^n/n! \approx \mathrm{e}^n$ smaller than the number of operations used by Kannan's algorithm.

4 Successive Minima and the Generalized Closest Vector Problem

In this section we describe the algorithms for the generalized closest vector problem and the shortest independent vectors problem. Since the basic ideas and proofs are almost identical to the ones presented in the previous section we will be brief. First, we need to generalize Lemma 2 and Lemma 3 appropriately.

Lemma 6. *Let* $\mathsf{L} \subset \mathbb{R}^m$ *be a lattice of rank* n *with dual HKZ-basis* $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$*, let* $\mathrm{A} \subset \mathbb{R}^m$ *be an affine subspace with* $\mathsf{L} \not\subset \mathrm{A}$*, and let* $\mathbf{t}$ *be a vector in* $\operatorname{span}(\mathsf{L})$*,* $\mathbf{t} = \sum_{j=1}^{n} q_j \mathbf{b}_j, q_j \in \mathbb{Q}$*. If* $\mathbf{u} = \sum_{j=1}^{n} c_j \mathbf{b}_j, c_j \in \mathbb{Z}$*, is a vector in* $\mathsf{L} \backslash \mathrm{A}$ *closest to* $\mathbf{t}$*, then*

$$|q_n - c_n| \leq 3/2 \cdot n.$$

To prove this lemma, first we prove two auxiliary lemmas.

Lemma 7. *Let* $\mathsf{L} \subset \mathbb{R}^m$ *be a lattice. If* A *is an affine subspace of* $\mathbb{R}^m$ *with* $\mathsf{L} \not\subset \mathrm{A}$*, then the affine space* $\mathrm{A}' = \mathrm{A} \cap \mathrm{span}(\mathsf{L})$ *has dimension at most* $n-1$*.*

Proof. Since $\mathsf{L} \not\subset \mathrm{A}$, we conclude that $\mathrm{span}(\mathsf{L}) \not\subset \mathrm{A}$. Hence the affine space $\mathrm{A}' = \mathrm{A} \cap \mathrm{span}(\mathsf{L})$ is a proper subset of $\mathrm{span}(\mathsf{L})$. This shows that the dimension of A' is at most $n-1$. □

Lemma 8. *Let* L *be a lattice with rank* n *and let* A' *be an affine subspace of* $\mathrm{span}(\mathsf{L})$ *with* $\dim(\mathrm{A}') < n$*. For all* $\mathbf{t} \in \mathrm{span}(\mathsf{L})$*, a closest vector* $\mathbf{u}$ *to* $\mathbf{t}$ *in the set* $\mathsf{L} \backslash \mathrm{A}'$ *has distance at most* $\mu(\mathsf{L}) + \lambda_n(\mathsf{L})$ *to* $\mathbf{t}$*.*

Proof. Let $\mathbf{v}$ be a closest vector to $\mathbf{t}$ in the lattice L. Let $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathsf{L}$ be linearly independent with $\|\mathbf{v}_i\| = \lambda_i(\mathsf{L})$. Since $\mathbf{v}_i, i = 1, \ldots, n$, are linearly independent, at least one vector in $\{\mathbf{v}, \mathbf{v}+\mathbf{v}_1, \ldots, \mathbf{v}+\mathbf{v}_n\}$ is not contained in A'. If $\mathbf{v} \notin \mathrm{A}'$ the lemma is clearly true. So assume $\mathbf{v}+\mathbf{v}_h, 1 \le h \le n$, is not contained in A'. Since $\|\mathbf{v}-\mathbf{t}\| \le \mu(\mathsf{L})$ by definition of the covering radius $\mu(\mathsf{L})$,

$$\|\mathbf{v}+\mathbf{v}_h-\mathbf{t}\| \le \|\mathbf{v}-\mathbf{t}\| + \|\mathbf{v}_h\| \le \mu(\mathsf{L}) + \lambda_h(\mathsf{L}) \le \mu(\mathsf{L}) + \lambda_n(\mathsf{L}),$$

□

Proof (of Lemma 6). Applying Lemma 7 shows that the dimension of $\mathrm{A}' = \mathrm{span}(\mathsf{L}) \cap \mathrm{A}$ is at most $n-1$. Since $\mathsf{L} \backslash \mathrm{A} = \mathsf{L} \backslash \mathrm{A}'$, Lemma 8 applied with lattice L, affine space A', and target vector $\mathbf{t}$ shows

$$\|\mathbf{u}-\mathbf{t}\| \le \mu(\mathsf{L}) + \lambda_n(\mathsf{L}). \tag{2}$$

Denote by $[\mathbf{b}_1^*, \ldots, \mathbf{b}_n^*]$ a dual basis to $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$. Using the Cauchy-Schwarz-inequality we obtain

$$|q_n - c_n| = \|\langle \mathbf{t}-\mathbf{u}, \mathbf{b}_1^* \rangle\| \le \|\mathbf{t}-\mathbf{u}\| \cdot \|\mathbf{b}_1^*\| \le (\mu(\mathsf{L}) + \lambda_n(\mathsf{L})) \cdot \lambda_1(\mathsf{L}^*),$$

where the last inequality follows from (2) and from $\|\mathbf{b}_1^*\| = \lambda_1(\mathsf{L}^*)$. Applying the transference bounds (Theorem 1) shows

$$|q_n - c_n| \le 3/2 \cdot n.$$

□

The next lemma generalizes Lemma 3. It is the key to the recursive step of our algorithm for the generalized closest vector problem.

Lemma 9. *Let* L *be a lattice of rank* n *with dual HKZ-basis* $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$*, let* $\mathrm{A} \subset \mathbb{R}^m$ *be an affine subspace with* $\mathsf{L} \not\subset \mathrm{A}$*, and let* $\mathbf{t}$ *be a vector in* $\mathrm{span}(\mathsf{L})$*. Fix an integer* c *and denote by* $\mathbf{w}$ *the orthogonal projection of* $\mathbf{t} - c \cdot \mathbf{b}_n, c \in \mathbb{Z}$*, onto* $\mathrm{span}(\mathsf{L}_{n-1})$*. The vector* $\mathbf{v} + c \cdot \mathbf{b}_n, \mathbf{v} \in \mathsf{L}_{n-1}$*, is a closest vector to* $\mathbf{t}$ *in* $\mathsf{L} \backslash \mathrm{A}$ *of the form* $\mathbf{u} + c \cdot \mathbf{b}_n, \mathbf{u} \in \mathsf{L}_{n-1}$*, iff* $\mathbf{v}$ *is a vector in* $\mathsf{L}_{n-1} \backslash (-c \cdot \mathbf{b}_n + \mathrm{A})$ *closest to* $\mathbf{w}$*.*

Proof. A vector $\mathbf{u} + c \cdot \mathbf{b}_n, \mathbf{u} \in \mathsf{L}_{n-1}$ is contained in $\mathsf{L} \backslash \mathrm{A}$, iff $\mathbf{u}$ is contained in $\mathsf{L}_{n-1} \backslash (-c \cdot \mathbf{b}_n + \mathrm{A})$. As in the proof for Lemma 3 we can now argue that for $\mathbf{v} \in \mathsf{L}_{n-1} \backslash (-c \cdot \mathbf{b}_n + \mathrm{A})$ we get

$$\begin{aligned} & \|\mathbf{v} + c \cdot \mathbf{b}_n - \mathbf{t}\| &= \min_{\mathbf{u} \in \mathsf{L}_{n-1} \backslash (-c \cdot \mathbf{b}_n + \mathrm{A})} \|\mathbf{u} + c \cdot \mathbf{b}_n - \mathbf{t}\| \\ \iff & \|\mathbf{v} - \mathbf{w}\| &= \min_{\mathbf{u} \in \mathsf{L}_{n-1} \backslash (-c \cdot \mathbf{b}_n + \mathrm{A})} \|\mathbf{u} - \mathbf{w}\|. \end{aligned}$$

This proves the lemma. □

Now we are in a position to state the algorithm solving the generalized closest vector problem. The input to the algorithm is a dual HKZ-basis $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ of L, an affine subspace $\mathrm{A} \subset \mathbb{R}^m$, and a target vector $\mathbf{t} = \sum_{j=1}^{n} q_j \mathbf{b}_j, q_j \in \mathbb{Q}$. To treat the case that $\mathsf{L} \subset \mathrm{A}$, we introduce a special symbol ω. The algorithms will output ω, iff $\mathsf{L} \subset \mathrm{A}$.

Algorithm Generalized Closest Vector

```
IF n = 1 THEN
    IF L ⊂ A THEN
        Return ω
    ELSE
        Compute the closest integer c to q_1
        IF c · b_1 ∉ A THEN
            Return c · b_1
        ELSE
            Compute the closest integer c' ≠ c to q_1 and return c' · b_1.
        END
    END
ELSE
    Set C := ∅.
    FOR c = −3/2 · n TO 3/2 · n DO
        Compute the orthogonal projection w of t − c · b_n onto span(L_{n−1}).
        Recursively call Algorithm Generalized Closest Vector with basis
        [b_1, ... , b_{n−1}], affine space −c · b_n + A, and target vector w.
        IF the recursive call returns a vector v ≠ ω THEN
            Set C := C ∪ {v + c · b_n}.
        END
        IF C ≠ ∅ THEN
            Compute and return a closest element in C to t.
        ELSE
            Return ω
        END
    END
```

Next we show the correctness of the algorithm and analyze its running time.

Lemma 10. *Assume the input to Algorithm Generalized Closest Vector is a dual HKZ-basis* $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$ *of the lattice* L*, a target vector* $\mathbf{t}$ *represented as* $\mathbf{t} = \sum_{j=1}^{n} q_j \mathbf{b}_j, q_j \in \mathbb{Q}$*, and an affine subspace* $\mathrm{A} \subset \mathbb{R}^m$*. Then Algorithm Generalized Closest Vector computes a closest vector in* $\mathsf{L} \backslash \mathrm{A}$ *to* $\mathbf{t}$*.*

Proof. We only show that for $n = 1$ the algorithm correctly computes a closest vector in $\mathsf{L} \backslash \mathrm{A}$. The remainder of the proof is almost identical to the proof of Lemma 4.

By Lemma 7, if $\mathsf{L} \not\subset \mathrm{A}$ then the dimension of $\mathrm{span}(\mathsf{L}) \cap \mathrm{A}$ is either 0 or the intersection $\mathrm{span}(\mathsf{L}) \cap \mathrm{A}$ is empty. Therefore, L and A either have empty intersection or they intersect in a single point. From this the correctness of Algorithm Generalized Closest Vector for lattices with rank 1 follows. □

Lemma 11. *Assume the lattice* $\mathsf{L} \subset \mathbb{R}^m$ *and the target vector* $\mathbf{t}$ *have representation size* s *with respect to the dual HKZ-basis* $[\mathbf{b}_1, \ldots, \mathbf{b}_n]$*. Also assume that the representation size of* A *is* s*. Then a vector in* $\mathsf{L} \backslash \mathrm{A}$ *closest to* $\mathbf{t}$ *can be computed with* $3^n \cdot n! \cdot m^{O(1)}$ *arithmetic operations on integers of size* $s^{O(1)}$*.*

Proof. The result follows exactly as Lemma 5 with the recurrence $T(n, m) = T(n-1, m) + m^{O(1)}$ replaced by the recurrence $T(n, m) = 3 \cdot n \cdot T(n-1, m) + m^{O(1)}$. □

Finally we obtain

Theorem 4. *Given a lattice* L *with rank* n*, an affine space* $\mathrm{A} \subset \mathbb{R}^m$*, and a vector* $\mathbf{t} \in \mathrm{span}(\mathsf{L})$*, all with representation size* s*. A closest vector to* $\mathbf{t}$ *in* $\mathsf{L} \backslash \mathrm{A}$ *can be computed with* $3^n \cdot n! \cdot m^{O(1)} + n^{n/2} \cdot s^{O(1)}$ *arithmetic operations on integers of size* $s^{O(1)}$*.*

The proof of this theorem is the same as the proof for Theorem 3. As mentioned in Section 1, Theorem 4 implies

Theorem 5. *Given a lattice* L *with rank* n *and representation size* s*. Linearly independent vectors* $\mathbf{v}_1, \ldots, \mathbf{v}_n \in \mathsf{L}$ *with* $\lambda_i(\mathsf{L}) = \|\mathbf{v}_i\|$ *can be computed with* $3^n \cdot n! \cdot m^{O(1)} + n^{n/2} \cdot s^{O(1)}$ *arithmetic operations on integers of size* $s^{O(1)}$*.*

Acknowledgment

I thank J.-P. Seifert for many useful and illuminating discussions.

References

1. M. Ajtai, "Generating Hard Instances of Lattice Problems", *Proc. 28th Symposium on Theory of Computing* 1996, pp. 99-108.
2. M. Ajtai, "Worst-Case Complexity, Average-Case Complexity and Lattice Problems", *Proc. International Congress of Mathematicians* 1998, Vol. III, pp. 421-428.
3. S. Arora, C. Lund, "Hardness of Approximations", in D. S. Hochbaum (ed.), *Approximation Algorithms for* NP*-Hard Problems*, PWS Publishing, 1997.
4. S. Arora, L. Babai, J. Stern, Z. Sweedyk, "The Hardness of Approximate Optima in Lattices, Codes, and Systems of Linear Equations", *Journal of Computer and System Sciences* Vol. 54, pp. 317-331, 1997.
5. M. Ajtai, C. Dwork "A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence", *Proc. 29th Symposium on Theory of Computing* 1997, pp. 284-293.
6. L. Babai, "On Lovasz Lattice Reduction Reduction and the Nearest Lattice Point Problem", *Combinatorica*, Vol. 6, pp. 1-6, 1986.
7. W. Banaszczyk, "New Bounds in Some Transference Theorems in the Geometry of Numbers", *Mathematische Annalen* Vol. 296, pp. 625-635, 1993.
8. J. Blömer, J.-P. Seifert, "The Complexity of Computing Short Linearly Independent Vectors and Short Bases in a Lattice", *Proc. 21stth Symposium on Theory of Computing* 1999, pp. 711-720.
9. J. W. S. Cassels, *An Introduction to the Geometry of Numbers*, Springer-Verlag, 1971.
10. J. Y. Cai, "A New Transference Theorem and Applications to Ajtai's Connection Factor", *Proc. 14th Conference on Computational Complexity* 1999, pp. 205-214.
11. J. Y. Cai, "Some Recent Progress on the Complexity of Lattice Problems", *Proc. 14th Conference on Computational Complexity* 1999, pp. 158-177.
12. J. Y. Cai, A. P. Nerurkar, "An Improved Worst-Case to Average-Case Reduction for Lattice Problems", *Proc. 38th Symposium on Foundations of Computer Science* 1997, pp. 468-477.
13. I. Dinur, G. Kindler, S. Safra, "Approximating CVP to Within Almost-Polynomial Factors is NP-Hard", *Proc. 39th Symposium on Foundations of Computer Science* 1998, pp 99-109.
14. O. Goldreich, S. Goldwasser, "On the Limits of Non-Approximability of Lattice Problems", *Proc. 30th Symposium on Theory of Computing* 1998, pp. 1-9.
15. B. Helfrich, "Algorithms to Construct Minkowski Reduced and Hermite Reduced Lattice Basis", *Theoretical Computer Science*, Vol. 41, pp. 125-139, 1985.
16. R. Kannan, "Minkowski's Convex Body Theorem and Integer Programming", *Mathematics of Operations Research*, Vol. 12, No. 3, pp. 415-440, 1987.
17. L. Lovasz, *An Algorithmic Theory of Graphs, Numbers and Convexity*, SIAM, 1986.
18. J. Lagarias, H. W. Lenstra, C.P. Schnorr, "Korkin-Zolotarev Bases and Successive Minima of a Lattice and its Reciprocal Lattice", *Combinatorica* Vol. 10, No. 4, pp. 333-348, 1990.
19. M. Pohst, "On the Computation of Lattice Vectors of Minimal Length, Successive Minima and Reduced Bases with Applications", *SIGSAM Bulletin* Vol. 15, No. 1, pp. 37-44, 1981.

Variable Independence, Quantifier Elimination, and Constraint Representations

Leonid Libkin[1,⋆]

Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974, USA.
libkin@research.bell-labs.com

Abstract. Whenever we have data represented by constraints (such as order, linear, polynomial, etc.), running time for many constraint processing algorithms can be considerably lowered if it is known that certain variables in those constraints are independent of each other. For example, when one deals with spatial and temporal databases given by constraints, the projection operation, which corresponds to quantifier elimination, is usually the costliest. Since the behavior of many quantifier elimination algorithms becomes worse as the dimension increases, eliminating certain variables from consideration helps speed up those algorithms.

While these observations have been made in the literature, it remained unknown when the problem of testing if certain variables are independent is decidable, and how to construct efficiently a new representation of a constraint-set in which those variables do not appear together in the same atomic constraints. Here we answer this question. We first consider a general condition that gives us decidability of variable independence; this condition is stated in terms of model-theoretic properties of the structures corresponding to constraint classes. We then show that this condition covers the domains most relevant to spatial and temporal applications. For some of these domains, including linear and polynomial constraints over the reals, we provide a uniform decision procedure which gives us tractability, and present a polynomial-time algorithm for producing nice constraint representations.

1 Introduction

We start with a simple example. Suppose we have a set $S \subseteq \mathbb{R}^2$ given by simple order-constraints $\varphi(x,y) = (0 < x < 1) \wedge (0 < y < 1)$. Suppose we want to find its projection on the x axis. This means writing the formula $\exists y\ \varphi(x,y)$ as a quantifier-free formula. This can be done, in general, because the theory of $\langle \mathbb{R}, <, (r)_{r\in\mathbb{R}} \rangle$ admits quantifier elimination. But in this particular case it is very easy to find a quantifier-free formula equivalent to $\exists y\ \varphi(x,y)$ using just standard rules for equivalence of first-order formulae:

$$\exists y\ \varphi(x,y) \ \leftrightarrow\ (0 < x < 1) \wedge \exists y\ (0 < y < 1) \ \leftrightarrow\ (0 < x < 1) \wedge \text{true} \ \leftrightarrow\ 0 < x < 1.$$

⋆ Part of this work was done while visiting INRIA.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 260–271, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Now notice that φ can be considered as a formula in the language of the real field $\langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$ whose theory also admits quantifier elimination. Suppose then that instead of φ, we are given an equivalent formula $\psi(x, y)$:

$$\begin{aligned} &\big((0 < x < 1) \wedge (0 < y < 1) \wedge (4x^2 - y - 1 \geq 0)\big) \\ \vee &\big((0 < x < 1) \wedge (0 < y < 1) \wedge (4x^2 - y - 1 \leq 0)\big). \end{aligned}$$

The first step of quantifier elimination for $\exists y\ \psi$ is easy, as we propagate $\exists y$ inside the disjunction. However, trying to find a quantifier-free equivalent for the first disjunct, that is, a formula equivalent to $\exists y\ \big((0 < x < 1) \wedge (0 < y < 1) \wedge (4x^2 - y - 1 \geq 0)\big)$, one immediately encounters obstacles. Unlike the earlier example, this one requires a bit of thought to come up with the answer $(0.5 \leq x < 1)$. Similarly, some work is needed to compute the answer $(0 < x \leq 1/\sqrt{2})$ for the second disjunct.

Why is it that the first quantifier-elimination procedure is completely elementary, and the second is not, even though both φ and ψ define the same set? The reason is that in the first representation of S, variables x and y are independent, that is, they do not appear in the same atomic formulae. This makes quantifier elimination easy. In the second case, x and y do appear together in the same term $x^2 - 4y - 1$, and this is what causes the problem.

This extremely simple observation can often make constraint processing easier. While it can conceivably be useful in various tasks such as more efficient variable elimination in constraint logic programming [8,12], here we concentrate on one application area, namely *constraint databases* [15,14] where it found its way into a practical system for querying spatio-temporal databases [9]. The main goal of constraint databases is to model infinite database objects, which arise in a variety of applications, for example, in Geographical Information Systems.

A particular constraint model is defined over a structure $\mathcal{M} = \langle U, \Omega \rangle$ (where U is the universe and Ω is the vocabulary) which is typically required to have quantifier elimination. Those considered most often in spatial application are the real field $\mathbf{R} = \langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$ and the real ordered group $\mathbf{R}_{\text{lin}} = \langle \mathbb{R}, +, -, 0, 1 < \rangle$, which give rise to polynomial and linear constraint databases, respectively. A constraint relation of arity n is simply a definable subset of U^n, that is, a set of tuples $\vec{a} \in U^n$ that satisfy a first-order formula. For the above structures, constraint relations are *semi-algebraic* sets for $\mathbf{R}$, and *semi-linear* sets for $\mathbf{R}_{\text{lin}}$ [2]. A constraint database is a finite set of constraint relations.

A standard constraint query language over $\mathcal{M}$ is $\text{FO} + \mathcal{M}$, that is, first-order logic in the language of $\mathcal{M}$ and symbols for relations in a constraint database. For example, if a database contains a single ternary symbol S, the query $\varphi(x) \equiv \exists u, v\ \forall y, z\ (S(x, y, z) \leftrightarrow z = u \cdot y + v)$ finds all a such that the intersection of S with the plane $x = a$ is a line. Note that if S is a semi-algebraic set, then so is $\varphi(S)$.

One of the standard database operations is projection. In the language of constraint processing, it corresponds to quantifier elimination. That is, given a quantifier-free formula $\varphi(y, x_1, \ldots, x_{n-1})$, one wishes to find a quantifier-free formula $\psi(\vec{x})$ equivalent to $\exists y\ \varphi(y, \vec{x})$. In many cases, the complexity of algorithms

to find such a ψ is of the form $O(N^{f(n)})$, where N is the size of the formula, and f is some function. For example, if one uses cylindrical algebraic decomposition [3] for the real field, f is $O(2^n)$. In general, even if better algorithms are available, the complexity of constraint processing often increases with dimension to such an extent that it becomes unmanageable for large datasets (see, e.g., [10]).

Assume now that $\vec{x}$ is split into two disjoint tuples $\vec{u}$ and $\vec{v}$ such that $(y, \vec{u})$ and $\vec{v}$ are independent, that is, they do not appear in the same atomic formulae. Then φ is equivalent to a formula of the form

$$\bigvee_{i=1}^{k} \alpha_i(y, \vec{u}) \wedge \beta_i(\vec{v}). \tag{1}$$

Therefore, the formula $\exists y\ \varphi$ is equivalent to

$$\bigvee_{i=1}^{k} (\exists y\ \alpha_i(y, \vec{u})) \wedge \beta_i(\vec{v}). \tag{2}$$

For a number of operations this is a significant improvement, as the exponent becomes lower. For example, in addition to quantifier elimination, data often has to be represented in a nice format (essentially, as union of cells [3]), and algorithms for doing this also benefit from reduction in the dimension [9,10].

Even though such a notion of independence may seem to be too much of a restriction, from the practical point of view it is sometimes necessary to insist on it, as the cost of general quantifier elimination and other operations could be prohibitively expensive. For example, the DEDALE constraint database system [9] requires that the projection operation only be applied when $\vec{u}$ consists of a single variable. Dealing with spatio-temporal applications, one often queries trajectories of objects, or cadastral (land-ownership) information. These are typically represented as objects in $\mathbb{R}^3$ given by formulae $\varphi(x, y, t)$. To be able to compute $\exists y\ \varphi(x, y, t)$, one approximates φ by a formula $\psi(x, y, t)$ which is a Boolean combination of formulae $\alpha_i(x, y)$ and $\beta_i(t)$. For trajectories, this amounts to saying that an object is in a given region during a given interval of time; thus, it is the information about the speed that is lost in order to have efficient query evaluation. As was further demonstrated in [10], the difference between the case when at most 2 variables are dependent, and that of 3 or more variables being dependent, is quite dramatic, in the case of linear and polynomial constraints.

What is missing, however, in this picture, is the ability to determine whether a given constraint representation of the data can be converted to the one in the right format, just as in our first example, $\psi(x, y)$ is equivalent to $\varphi(x, y)$, in which variables x and y are independent. It was claimed in [5] that such a procedure exists for linear constraints, and then [10] gave a simpler algorithm. However, [16] then showed that both claims were incorrect. It was thus not known if variable independence can be tested for relevant classes of constraints.

Our main goal here is to show that variable independence can be tested for many classes of constraints, and that algorithms for converting a given formula into a one in the right form can be obtained. Moreover, those algorithms often

work in time polynomial in the size of the formula (assuming the total number of variables is fixed). Among structures for which we prove such results are the real ordered group, the real field, as well as $\langle \mathbb{Z}, +, 0, 1, < \rangle$ extended with all the relations $x = y (\text{mod } k)$, $k > 1$ (which is used in temporal applications). Even if those algorithms are relatively expensive, it is worth putting data in a nice format for two reasons. First, such an algorithm works only once, and then the data is repeatedly queried by different queries, which can be evaluated faster. Secondly, some queries are known to preserve variable independence; hence, this information can be used for further processing the query output.

Organization In Section 2, we define the notion of variable independence, and more generally, the notion $\varphi \sim P$ of a formula φ respecting a certain partition P of its free variables. Then, in Section 3, we discuss requirements on the theory of $\mathcal{M}$ that guarantee decidability of this notion, as well as the existence of an algorithm that converts a given formula into a one in the right shape. In Section 4, we discuss specific classes of structures and derive some complexity bounds. In particular, we look at *o-minimal* structures [23] (which include linear and polynomial constraints over the reals) and give a *uniform* decision procedure. This procedure gives us tractability, and we also show how to find an equivalent formula in the right shape in polynomial time. All proofs are only sketched here; complete proofs are in the full version [17].

2 Notations

All the definitions can be stated for arbitrary first-order structures, although for the algorithmic considerations we shall require at least decidability of the theory, and often quantifier elimination.

Given a structure $\mathcal{M} = \langle U, \Omega \rangle$ (where U is a set always assumed to be infinite, and Ω can contain predicate, function, and constant symbols, and is always assumed to be a recursive set), we say that the theory of $\mathcal{M}$ is decidable if for every first-order sentence Φ in the language of $\mathcal{M}$ it decidable if $\mathcal{M} \models \Phi$. We say that $\mathcal{M}$ admits (effective) quantifier elimination if for every formula $\varphi(\vec{x})$ in the language of $\mathcal{M}$, there exists (and can be effectively found) a quantifier-free formula $\psi(\vec{x})$ such that $\mathcal{M} \models \forall \vec{x}\ \varphi(\vec{x}) \leftrightarrow \psi(\vec{x})$.

Given a formula $\varphi(\vec{x}, \vec{y})$ in the language of $\mathcal{M}$, with $\vec{x}$ of length n and $\vec{y}$ of length m, and $\vec{a} \in U^n$, we write $\varphi(\vec{a}, \mathcal{M})$ for the set $\{\vec{b} \in U^m \mid \mathcal{M} \models \varphi(\vec{a}, \vec{b})\}$. In the absence of variables $\vec{x}$ we write $\varphi(\mathcal{M})$ for $\{\vec{b} \mid \mathcal{M} \models \varphi(\vec{b})\}$. Sets of the form $\varphi(\mathcal{M})$ are called *definable*. A function $f : U^n \to U^m$ is definable if its graph $\{(\vec{a}, \vec{b}) \in U^{n+m} \mid \vec{b} = f(\vec{a})\}$ is a definable set.

Given a tuple of variables $\vec{x} = (x_1, \ldots, x_n)$ and a partition $P = \{B_1, \ldots, B_m\}$ on $\{1, \ldots, n\}$, we let $\vec{x}_{B_i}$ stand for the subtuple of $\vec{x}$ consisting of the x_js with $j \in B_i$. For a formula $\varphi(x_1, \ldots, x_n)$, we then say that φ *respects the partition* P (over $\mathcal{M}$) if φ is equivalent to a Boolean combination of formulae each having its free variables among $\vec{x}_{B_i}$ for some $i \leq k$. This will be written as $\varphi \sim_{\mathcal{M}} P$, or just $\varphi \sim P$ if $\mathcal{M}$ is clear from the context.

In other words (by putting a Boolean combination into DNF), $\varphi \sim_{\mathcal{M}} P$ if there exists a family of formulae $\alpha_j^i(\vec{x}_{B_i})$, $i = 1, \ldots, m$, $j = 1, \ldots, k$, such that

$$(*) \qquad \mathcal{M} \models \varphi(\vec{x}) \leftrightarrow \bigvee_{j=1}^{k} (\alpha_j^1(\vec{x}_{B_1}) \wedge \ldots \wedge \alpha_j^m(\vec{x}_{B_m}))$$

When $\mathcal{M}$ has quantifier elimination, all α_j^is are quantifier free. In fact, under the quantifier-elimination assumption, the definition of $\varphi \sim_{\mathcal{M}} P$ can be restated as the equivalence of φ to a quantifier-free formula ψ such that every atomic subformula of ψ uses variables from only one block of P.

We say that in φ, two variables x_i and x_j are *independent* if there exists a partition P such that $\varphi \sim_{\mathcal{M}} P$, and x_i and x_j are in two different blocks of P. Equivalently, x_i and x_j are independent if there exists a partition $P = (\vec{y}, \vec{z})$ of $\vec{x}$ such that $\varphi \sim_{\mathcal{M}} P$, x_i is in $\vec{y}$ and x_j is in $\vec{z}$. (When convenient notationally, we identify partitions on the indices of variables and variables themselves.)

Structures. After presenting a general decidability result, we shall deal with several important classes of structures. Two of them were mentioned already: the real ordered group $\mathbf{R}_{\text{lin}} = \langle \mathbb{R}, +, -, 0, 1, < \rangle$ and the real field $\mathbf{R} = \langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$, corresponding to linear and polynomial constraints over the reals. Some of the results for these structures extend to a larger class of *o-minimal* structures: $\mathcal{M} = \langle U, \Omega \rangle$ is called o-minimal [19,23] if one of the symbols in Ω is $<$, interpreted as a linear order on U, and every definable subset of U, $\{a \mid \mathcal{M} \models \varphi(a)\}$, is a finite union of points and open intervals. Both $\mathbf{R}_{\text{lin}}$ and $\mathbf{R}$ have quantifier elimination (by Fourier elimination [25], and Tarski's theorem [2,3], respectively), which easily implies that they are o-minimal. The exponential field $\langle \mathbb{R}, +, \cdot, e^x \rangle$ is an example of a structure which is o-minimal [24] but does not have quantifier elimination [22]. For other o-minimal structures on the reals, see [23].

We shall deal with some structures on the integers. Of most interest to us is $\mathcal{Z}_0 = \langle \mathbb{Z}, +, -, 0, 1, <, (\equiv_k)_{k>1} \rangle$ where $n \equiv_k m$ iff $n = m (\text{mod } k)$. This structure corresponds to constraints given by linear repeating points, which are used for modeling temporal databases [13]. The structure $\mathcal{Z}_0$ admits effective quantifier elimination, and its theory is decidable [7].

3 General Conditions for Deciding Variable Independence

Given a structure $\mathcal{M}$, we consider two problems. The *variable independence problem* $\text{VI}_{\mathcal{M}}(\varphi, x_i, x_j)$ is to decide, for $\varphi(x_1, \ldots, x_n)$ in the language of $\mathcal{M}$, if x_i and x_j are independent. The *variable partition problem* $\text{VP}_{\mathcal{M}}(\varphi, P)$ is to decide, for a given formula $\varphi(x_1, \ldots, x_n)$ and a partition P on $\{1, \ldots, n\}$, if $\varphi \sim_{\mathcal{M}} P$.

Note that the variable independence problem is a special case of the variable partition problem, as to solve the former, one needs to solve the latter for some partition $P = (B_1, B_2)$ with $i \in B_1$ and $j \in B_2$.

The above problems are just decision problems, but if the theory of $\mathcal{M}$ is decidable, and the answer to $\mathrm{VP}_{\mathcal{M}}(\varphi, P)$ is 'yes', one can effectively find a representation in the form $(*)$, simply by enumerating all the formulae $\langle\psi(\vec{x})\rangle_i$ which are Boolean combinations of formulae having free variables from at most one block of P, and then checking if $\mathcal{M} \models \forall\vec{x}\ (\varphi(\vec{x}) \leftrightarrow \psi_i(\vec{x}))$. Since $\varphi \sim_{\mathcal{M}} P$, for some finite i, we get a positive answer. In many interesting cases, we shall see better algorithms for finding representation $(*)$ than simple enumeration.

The first easy result shows that the problems $\mathrm{VI}_{\mathcal{M}}(\varphi, x_i, x_j)$ and $\mathrm{VP}_{\mathcal{M}}(\varphi, P)$ are equivalent; this allows us to deal then only with two-block partitions.

Lemma 1. *For any $\mathcal{M}$, the variable independence problem is decidable over $\mathcal{M}$ iff the variable partition problem is decidable over $\mathcal{M}$.* □

Next, we discuss conditions for decidability of the variable independence problem. It is clear that one needs decidability of the theory of $\mathcal{M}$. However, decidability alone (and even effective quantifier elimination) are not sufficient.

Proposition 1. *a) If the theory of $\mathcal{M}$ is undecidable, then the variable independence problem is undecidable over $\mathcal{M}$.*

b) There exists a structure $\mathcal{M}$ with a decidable theory and effective quantifier elimination such that the variable independence problem is undecidable over $\mathcal{M}$.

Proof sketch. a) If Φ is a sentence and $\varphi(x, y)$ is $(x = y) \wedge \neg\Phi$, then x and y are independent in φ iff $\mathcal{M} \models \Phi$.

b) An example is provided by the theory of traces from [21]. Let U be a union of three disjoint sets: descriptions of Turing machines, input words, and traces, or partial computations of machines on input words, all appropriately coded as strings. Let Ω contain a constant symbol for every element of U, and a single ternary predicate $P(m, w, t)$ saying that t is a trace of the machine m on the input word w. This signature can be expanded by finitely many symbols so that the expanded model has effective quantifier elimination.

Now fix a Turing machine m_0 and an input word w_0 and consider the formula $\varphi(t, t') = (P(m_0, w_0, t) \wedge (t = t'))$. We then show that t and t' are independent iff m_0 halts on w_0. □

The proof of Proposition 1, b), shows that it is essential to be able to decide finiteness in order to decide $\mathrm{VI}(\varphi, x_i, x_j)$ (as it is the finiteness of the number of traces that turns out to be equivalent to variable independence). Recall that a formula $\varphi(x)$ is *algebraic* if $\varphi(\mathcal{M})$ is finite. We say that there is an *effective test for algebraicity* in $\mathcal{M}$ if for every $\varphi(x)$ in the language of $\mathcal{M}$, it is decidable if φ is algebraic. Note that this somewhat technical notion will trivially hold for most relevant classes of constraints.

While the notion of variable independence is needed in the context of constraint databases, for finite relational structures it is assumed to be meaningless as every tuple is represented as a conjunction of constraints of the form $x_i = c_i$, where c_is are constants. For example, the graph $\{(1, 2), (3, 4)\}$ is given by the

formula $((x = 1) \wedge (y = 2)) \vee ((x = 3) \wedge (y = 4))$. Clearly, variables x and y are independent.

However, over arbitrary structures, not every finite definable set would satisfy the variable independence condition. To see this, let $\mathcal{M} = \langle \mathbb{N}, C, E \rangle$, where C is a unary relation interpreted as $\{1, 2\}$ and E is a binary relation symbol interpreted as $\{(1, 2), (2, 1)\}$. A routine argument shows that this $\mathcal{M}$ has quantifier elimination, decidable theory, and there is a test for algebraicity. The formula $\varphi(x, y) \equiv E(x, y)$ then defines a finite set, but variables x and y are not independent: this is because the only definable proper subsets of $\mathbb{N}$ are $\{1, 2\}$ and $\mathbb{N} - \{1, 2\}$, and no Boolean combination of those gives us E. As another example, consider the field of complex numbers, whose theory is decidable and has quantifier elimination [18]. Let $\varphi(x, y) = (x^2 + 1 = 0) \wedge (y^2 + 1 = 0) \wedge (x + y = 0)$. It defines the finite set $\{(i, -i), (-i, i)\}$ but nevertheless x and y are not independent (since i is not definable).

To avoid similar situations, we impose an extra condition on a structure, again, well known in model theory [4,11]. We say that $\mathcal{M}$ has *definable Skolem functions* if for every formula $\varphi(\vec{x}, \vec{y})$ there exists a definable function $f_\varphi(\vec{x})$ with the property that $\mathcal{M} \models \forall \vec{x}\ (\exists \vec{y}\ \varphi(\vec{x}, \vec{y}) \rightarrow \varphi(\vec{x}, f_\varphi(\vec{x})))$. In other words, $f_\varphi(\vec{a})$ is an element of $\varphi(\vec{a}, \mathcal{M})$, assuming $\varphi(\vec{a}, \mathcal{M})$ is not empty. We say that a Skolem function f_φ is *invariant* [18], if $\varphi(\vec{a}_1, \mathcal{M}) = \varphi(\vec{a}_2, \mathcal{M})$ implies $f_\varphi(\vec{a}_1) = f_\varphi(\vec{a}_2)$. If the existence of such a Skolem function can be guaranteed for every φ, we say that $\mathcal{M}$ has definable invariant Skolem functions.

Theorem 1. *Assume that $\mathcal{M}$ has the following properties:*

(a) its theory is decidable;
(b) $\mathcal{M}$ has effective test for algebraicity; and
(c) $\mathcal{M}$ has definable invariant Skolem functions.

Then the variable partition and independence problems are decidable over $\mathcal{M}$.

Proof sketch. We consider the case of two block partitions; that is, deciding if a formula $\varphi(\vec{x}, \vec{y})$ respects the partition P with blocks $\vec{x}$ and $\vec{y}$. Let $\vec{x}$ have length n and $\vec{y}$ have length l. Define an equivalence relation on U^n by

$$\vec{a}_1 \equiv \vec{a}_2 \quad \text{iff} \quad \varphi(\vec{a}_1, \mathcal{M}) = \varphi(\vec{a}_2, \mathcal{M}).$$

Lemma 2. *For φ, P and $\equiv$ as above, $\varphi \sim_{\mathcal{M}} P$ iff $\equiv$ has finitely many equivalence classes.* □

Using this and the assumptions on $\mathcal{M}$, we show how to define a formula $\chi(\vec{x})$ finding a set of representatives of the equivalence classes of $\equiv$; then again using the assumptions on $\mathcal{M}$ we show that it is decidable if $\chi(\mathcal{M})$ is finite. □

The proof of Theorem 1 gives an explicit construction for a formula witnessing $\varphi \sim_{\mathcal{M}} P$, where P has two blocks. We now extend it to arbitrary partitions.

Let $\varphi(x_1, \ldots, x_n)$ be given, and let $B \subset \{1, \ldots, n\}$. Let $card(B) = k$. For $\vec{a} \in U^k$, by $\varphi_B(\vec{a}, \mathcal{M})$ we denote the set of $\vec{b} \in U^{n-k}$ such that $\varphi(\vec{c})$ holds, where

$\vec{c}$ is obtained from $\vec{a}$ and $\vec{b}$ by putting their elements in the appropriate position, $\vec{a}$ being in the positions specified by B. For example, if $n = 4$, $B = \{2, 4\}$, and $\vec{a} = (a_1, a_2)$, $\vec{b} = (b_1, b_2)$, then $\vec{c}$ is (b_1, a_1, b_2, a_2). Formally, for $i \in [1, n]$, let k_1 be the number of $j \in B$ with $j \leq i$, and k_2 be the number of $j \notin B$ with $j \leq i$. Then c_i is a_{k_1} if $i \in B$, and b_{k_2}, if $i \notin B$.

We use the notation

$$\vec{a}_1 \equiv^{\varphi}_{B_i} \vec{a}_2 \quad \text{iff} \quad \varphi_{B_i}(\vec{a}_1, \mathcal{M}) = \varphi_{B_i}(\vec{a}_2, \mathcal{M}).$$

We now obtain the following characterization of $\mathrm{VP}_{\mathcal{M}}(\varphi, P)$.

Corollary 1. *Let $\mathcal{M}$ be as in Theorem 1, and let $\varphi(x_1, \ldots, x_n)$ and a partition $P = (B_1, \ldots, B_m)$ on $\{1, \ldots, n\}$ be given. Then:*

1. *For each $i \leq m$, it is decidable if the equivalence relation $\equiv^{\varphi}_{B_i}$ has finitely many equivalence classes. Furthermore, $\varphi \sim_{\mathcal{M}} P$ iff each $\equiv^{\varphi}_{B_i}$ has finitely many classes.*
2. *If $\varphi \sim_{\mathcal{M}} P$, then one can further effectively find integers $N_1, \ldots, N_m > 0$ and formulae $\alpha^i_j(\vec{x}_{B_i})$, $i = 1, \ldots, m$, $j = 1, \ldots, N_i$, such that $\equiv^{\varphi}_{B_i}$ has N_i equivalence classes, which are definable by the formulae $\alpha^i_j(\vec{x}_{B_i})$, $j \leq N_i$. Furthermore,*

$$\mathcal{M} \models \forall \vec{x} \left(\varphi(\vec{x}) \leftrightarrow \bigvee_{(j_1, \ldots, j_m) \in K} \alpha^1_{j_1}(\vec{x}_{B_1}) \wedge \ldots \wedge \alpha^m_{j_m}(\vec{x}_{B_m}) \right)$$

where

$$K = \{(j_1, \ldots, j_m) \mid \mathcal{M} \models \exists \vec{x} \left(\alpha^1_{j_1}(\vec{x}_{B_1}) \wedge \ldots \wedge \alpha^m_{j_m}(\vec{x}_{B_m}) \wedge \varphi(\vec{x}) \right) \}.$$

4 Decidability for Specific Classes of Constraints

The general decidability result can be applied to a variety of structures, most notably, those that we listed earlier as the ones particularly relevant to constraint database applications (especially to spatial and temporal databases). In fact, the problem will be shown to be decidable for linear constraints over the rationals and the reals (this corresponds to structures $\langle \mathbb{Q}, +, -, 0, 1, < \rangle$ and $\mathbf{R}_{\mathrm{lin}}$), polynomial constraints over the reals ($\mathbf{R}$), and linear repeating points [13] ($\mathcal{Z}_0$).

4.1 Constraints on the Integers

Here the result follows easily form Theorem 1.

Proposition 2. *Let $\mathcal{M}$ be $\langle \mathbb{N}, <, \ldots \rangle$ or $\langle \mathbb{Z}, <, \ldots \rangle$, and let its theory be decidable. Assume, in the latter case, that there is at least one definable constant in $\mathcal{M}$. Then the variable partition and independence problems are decidable over $\mathcal{M}$.* □

Corollary 2. *The variable partition problem is decidable over $\mathcal{Z}_0 = \langle \mathbb{Z}, +, -, 0, 1, <, (\equiv_k)_{k>1} \rangle$.* □

4.2 Linear and Polynomial Constraints over the Reals

The linear constraints over the reals (corresponding to the structure $\mathbf{R}_{\text{lin}} = \langle \mathbb{R}, +, -, 0, 1, < \rangle$) and the polynomial constraints over the reals (corresponding to $\mathbf{R} = \langle \mathbb{R}, +, \cdot, 0, 1, < \rangle$) are the most useful constraints for spatial and spatio-temporal applications, where the problem of variable independence originated, and where variable independence is used in system prototypes. We thus concentrate on these constraints.

In many cases, however, we can state the results in greater generality using the concept of o-minimality (cf. section 2).

It is known that every o-minimal expansion of the $\mathbf{R}_{\text{lin}}$ has definable invariant Skolem functions [18,23]. Since every definable subset of U is a finite union of points and open intervals, one can test algebraicity, assuming that the order is dense: given $\varphi(x)$, the sentence $\exists u \exists v \forall x\ (u < x < v \to \varphi(x))$ tests if $\varphi(\mathcal{M})$ is infinite. This shows

Corollary 3. *Let $\mathcal{M} = \langle \mathbb{R}, +, 0, 1, <, \ldots \rangle$ be o-minimal, and have a decidable theory. Then the variable partition and independence problems are decidable over $\mathcal{M}$. In particular, these problems are decidable over $\mathbf{R}_{\text{lin}}$ and $\mathbf{R}$.* □

Since $\langle \mathbb{Q}, +, -, 0, 1, < \rangle$ is elementarily equivalent to $\mathbf{R}_{\text{lin}}$, we conclude that the variable partition problem is decidable over it, too.

Uniform Decidability and Complexity Bounds Our next goal is to present a *uniform* procedure for solving the problem $\text{VI}_{\mathcal{M}}(\varphi, P)$. More precisely, we say that the variable independence problem is *uniformly decidable* over $\mathcal{M}$ if the theory of $\mathcal{M}$ is decidable, and for every partition P on $\{1, \ldots, n\}$, there exists a single sentence Φ_P in the language of $\mathcal{M}$ expanded with an n-ary relation symbol S such that for any formula $\varphi(x_1, \ldots, x_n)$,

$$\varphi \sim_{\mathcal{M}} P \quad \text{iff} \quad (\mathcal{M}, \varphi(\mathcal{M})) \models \Phi_P.$$

Here $(\mathcal{M}, \varphi(\mathcal{M}))$ is the expansion of $\mathcal{M}$ where the new symbol S is interpreted as $\{\vec{a} \mid \mathcal{M} \models \varphi(\vec{a})\}$. Note that the decidability of the theory of $\mathcal{M}$ implies that $(\mathcal{M}, \varphi(\mathcal{M})) \models \Phi_P$ is decidable.

Proposition 3. *Let $\mathcal{M} = \langle \mathbb{R}, +, 0, 1, <, \ldots \rangle$ be o-minimal and have a decidable theory. Then the variable independence problem and partition problems are uniformly decidable over $\mathcal{M}$.*

Proof sketch. We show explicitly how to construct invariant Skolem function for a given equivalence relation. Given a (definable) set Y of representatives of a definable equivalence relation, its finiteness is tested as follows: Let X be the set of all coordinates of elements of Y. Since this is a definable set, it is finite iff it does not contain an open interval (by o-minimality). This condition can be tested by a sentence in the language of $\mathcal{M}$. □

Proposition 3 implies that the variable independence problem is uniformly decidable over $\mathbf{R}_{\text{lin}}$ and $\mathbf{R}$. The main application of this result is in establishing complexity bounds.

Since $\mathbf{R}$ admits quantifier elimination, every semi-algebraic set is given by a Boolean combination of polynomial inequalities. Thus, a standard way to represent a semi-algebraic set in $\mathbb{R}^n$ [1,3,20] is by specifying a collection of polynomials $p_1, \ldots, p_k \in \mathbb{Z}[x_1, \ldots, x_n]$, and defining a set X as a Boolean combination of sets of the form $\{\vec{a} \mid p_i(\vec{a}) \ \theta \ 0\}$, where θ is either $=$ or $>$. Here $\mathbb{Z}[x_1, \ldots, x_n]$, as usual, is the set of all polynomials in n variables with coefficients from $\mathbb{Z}$. One can use coefficients from $\mathbb{Q}$ as well, but this would not affect the class of definable sets.

Thus, when we study complexity of $\text{VP}_{\mathbf{R}}(\varphi, P)$, we assume that φ is given a Boolean combination of polynomial equalities and inequalities, with all polynomials having integer coefficients. The size of the input formula is then defined in a standard way, assuming that all integer coefficients are given in binary. All the above applies to semi-linear sets (that is, sets definable over $\mathbf{R}_{\text{lin}}$); we just restrict our attention to polynomials of degree 1.

Corollary 4. *Let $\mathcal{M}$ be $\mathbf{R}_{\text{lin}}$ or $\mathbf{R}$. Let P be a fixed partition on $\{1, \ldots, n\}$. Then, for a semi-algebraic (semi-linear) set given by a Boolean combination $\varphi(\vec{x})$ of polynomial inequalities (of degree 1), the problem $\text{VI}_{\mathcal{M}}(\varphi, P)$ is solvable in time polynomial in the size of φ.*

Proof sketch. This follows from Proposition 3 and complexity bounds on quantifier elimination [1,20]. □

Another reason to consider the uniform decision procedure for variable independence is that it gives us a test for variable independence under some transformations. For example, linear coordinate change in general would destroy variable independence, although it has relatively little effect on shapes on objects in $\mathbb{R}^n$. Consider, for example, the following version of the variable independence problem $\text{LVI}(X, x_i, x_j)$: Given a semi-algebraic set $X \subseteq \mathbb{R}^n$ (defined by a formula over $\mathbf{R}$), is there a linear change of coordinates such that in the new coordinate system, variables x_i and x_j are independent?

The general decision procedure of Theorem 1 does not give us a decision procedure for LVI. However, using uniformity, we easily obtain:

Corollary 5. *The problem $\text{LVI}(X, x_i, x_j)$ is decidable.* □

It turns out that not only the decision part of $\text{VI}_{\mathcal{M}}(\varphi, P)$ and $\text{VP}_{\mathcal{M}}(\varphi, P)$ can be solved in polynomial time for a fixed P over $\mathbf{R}_{\text{lin}}$ and $\mathbf{R}$, but there is also a polynomial time algorithm for finding a formula equivalent to φ that witnesses $\varphi \sim_{\mathcal{M}} P$.

Theorem 2. *1. Given $n > 1$, and a partition $P = (B_1, \ldots, B_k)$ on $\{1, \ldots, n\}$, there exists an algorithm that, for every semi-algebraic set given by a formula $\varphi(x_1, \ldots, x_n)$ which is a Boolean combination of polynomial equalities and inequalities, tests if $\varphi \sim_{\mathcal{M}} P$, and in the case of the positive answer, computes quantifier-free formulae $\alpha^i_j(\vec{x}_{B_i})$ such that each $\alpha^i_j(\vec{x}_{B_i})$ is*

a Boolean combination of polynomial (in)equalities (where polynomials depend only on $\vec{x}_{B_i}$ and all coefficients are integers), and $\varphi(\vec{x})$ is equivalent to $\bigvee_j \bigwedge_i \alpha^i_j(\vec{x}_{B_i})$. Moreover the algorithm works in time polynomial in the size of φ.

2. *The same statement is true when on replaces semi-algebraic by semi-linear, and all polynomials are of degree* 1.

Proof combines Corollary 1, uniform decidability (Proposition 3), complexity bounds for quantifier elimination [1,20] and, for 1), algorithms for polynomial root isolation [6]. □

In the full version, we also consider the most typical case of spatio-temporal applications: sets in $\mathbb{R}^3$ given by formulae $\varphi(x, y, t)$, where x, y describe the spatial component and t describes the temporal component. For this case, we present an algorithm based on cylindrical algebraic decomposition [3] for faster testing of variable independence.

5 Conclusion

We looked at the problem of deciding, for a set represented by a collection of constraints, whether some variables in those constraints are independent of each other. Knowing this can considerably improve the running time of several constraint processing algorithms, in particular, quantifier elimination. The problem originated in the field of spatio-temporal databases represented by constraints (linear or polynomial over the reals, for example); it was demonstrated that on large datasets, reasonable performance can only be achieved if variables comprise small independent groups. It had not been known, however, if such independence conditions are decidable.

Here we showed that these conditions are decidable for a large class of constraints, including those relevant to spatial and temporal applications. Moreover, for linear and polynomial constraints over the reals, we gave a uniform decision procedure that implies tractability, and we showed that a given constraint set can be converted into one in a nice shape in polynomial time, too.

Acknowledgements

I thank Stavros Cosmadakis and Gabi Kuper for bringing the problem to my attention, and Michael Benedikt and Luc Segoufin for their comments.

References

1. S. Basu. New results on quantifier elimination over real closed fields and applications to constraint databases. *Journal of the ACM*, 46 (1999), 537–555.
2. J. Bochnak, M. Coste, M.-F. Roy. *Real Algebraic Geometry.* Springer Verlag, 1998.
3. B.F. Caviness and J.R. Johnson, Eds. *Quantifier Elimination and Cylindrical Algebraic Decomposition.* Springer Verlag, 1998.

4. C.C. Chang and H.J. Keisler. *Model Theory.* North Holland, 1990.
5. J. Chomicki, D. Goldin and G. Kuper. Variable independence and aggregation closure. In *PODS'96*, pages 40–48.
6. G. E. Collins and R. Loos. Real zeros of polynomials. In *Computer Algebra: Symbolic and Algebraic Computation*, Springer-Verlag, 1983, pages 83–94.
7. H. B. Enderton. *A Mathematical Introduction to Logic.* Academic Press, New York, 1972.
8. A. Fordan and R. Yap. Early projection in CLP(R). In *CP'98*, pages 177–191.
9. S. Grumbach, P. Rigaux, L. Segoufin. The DEDALE system for complex spatial queries. In *SIGMOD'98*, pages 213–224.
10. S. Grumbach, P. Rigaux, L. Segoufin. On the orthographic dimension of constraint databases. In *ICDT'99*, pages 199–216.
11. W. Hodges. *Model Theory.* Cambridge, 1993.
12. J.-L. Imbert. Redundancy, variable elimination and linear disequations. In *SLP'94*, pages 139–153.
13. F. Kabanza, J.-M. Stevenne and P. Wolper. Handling infinite temporal data. *Journal of Computer and System Sciences*, 51 (1995), 3–17.
14. P. Kanellakis, G. Kuper, and P. Revesz. Constraint query languages. *Journal of Computer and System Sciences*, 51 (1995), 26–52. Extended abstract in *PODS'90*, pages 299–313.
15. G. Kuper, L. Libkin and J. Paredaens, eds. *Constraint Databases.* Springer Verlag, 2000.
16. L. Libkin. Some remarks on variable independence, closure, and orthographic dimension in constraint databases. *SIGMOD Record* 28(4) (1999), 24–28.
17. L. Libkin. Variable independence, quantifier elimination, and constraint representations. Bell Labs Technical Memo, 1998.
18. D. Marker, M. Messmer and A. Pillay. *Model Theory of Fields.* Springer Verlag, 1996.
19. A. Pillay, C. Steinhorn. Definable sets in ordered structures. III. *Trans. AMS* 309 (1988), 469–476.
20. J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. *J. Symb. Comp.* 13 (1992), 255–352.
21. A. Stolboushkin and M. Tsaitlin. Finite queries do not have effective syntax. *Information and Computation*, 153 (1999), 99–116.
22. L. van den Dries. Remarks on Tarski's problem concerning (R,+,*,exp). In *Logic Colloquium'82*, North Holland, 1984, pages 97–121.
23. L. van den Dries. *Tame Topology and O-Minimal Structures.* Cambridge, 1998.
24. A.J. Wilkie. Model completeness results for expansions of the ordered field of real numbers by restricted Pfaffian functions and the exponential function. *J. Amer. Math. Soc.* 9 (1996), 1051–1094.
25. G.M. Ziegler. *Lectures on Polytopes.* Springer-Verlag, 1994.

Constraint Satisfaction Problems and Finite Algebras

Andrei A. Bulatov[1], Andrei A. Krokhin[1], and Peter Jeavons[2]

[1] Department of Algebra and Discrete Mathematics
Ural State University, Ekaterinburg, Russia
Andrei.{Bulatov,Krokhin}@usu.ru

[2] Computing Laboratory
University of Oxford, UK
Peter.Jeavons@comlab.ox.ac.uk

Abstract. Many natural combinatorial problems can be expressed as constraint satisfaction problems. This class of problems is known to be **NP**-complete in general, but certain restrictions on the form of the constraints can ensure tractability. In this paper we show that any restricted set of constraint types can be associated with a finite universal algebra. We explore how the computational complexity of a restricted constraint satisfaction problem is connected to properties of the corresponding algebra. Using these results we exhibit a common structural property of all known intractable constraint satisfaction problems. Finally, we classify all finite strictly simple surjective algebras with respect to tractability. The result is a dichotomy theorem which significantly generalises Schaefer's dichotomy for the Generalised Satisfiability problem.

1 Introduction

The constraint satisfaction problem provides a framework in which it is possible to express, in a natural way, a wide variety of combinatorial problems [1,12,14]. The aim in a constraint satisfaction problem is to find an assignment of values to a given set of variables, subject to constraints on the values which can be assigned simultaneously to certain specified subsets of the variables.

The mathematical framework used to describe constraint satisfaction problems has strong links with several other areas of computer science and mathematics. For example, links with relational database theory [1,5] and with some notions of group theory [3] and universal algebra [8] have been investigated. There is a survey of these results in [15].

The constraint satisfaction problem is known to be **NP**-complete in general [11]. However, certain restrictions on the form of the constraints can ensure tractability [3,6,7,9,10]. In [9], it is proved that the time complexity of a problem involving a restricted set of constraint relations is determined by certain algebraic invariance properties of the constraint relations: namely, by the *clone of polymorphisms* of these relations. This result transforms the search for new tractable constraint types to the search for clones leading to tractability.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 272–282, 2000.
© Springer-Verlag Berlin Heidelberg 2000

In this paper we extend this result by exploring further the fundamental connection between constraint relations, clones of operations, and finite algebras. The motivation for this research is that by characterising sets of relations in terms of their algebraic properties we can use powerful algebraic tools to analyse the complexity of the corresponding problems. Furthermore, in many cases the algebraic characterisation is simpler than an explicit description. As an example of the power of the algebraic approach we show that Schaefer's theorem [17] concerning the complexity of the GENERALISED SATISFIABILITY problem can be expressed very simply as a classification of two-element algebras with respect to tractability.

The paper is organised as follows. In Section 2 we give basic definitions relating to the constraint satissfaction problem, clones of polymorphisms, and finite algebras, and we define the notion of a tractable algebra. In Section 3 we prove that, for our purposes, it suffices to consider idempotent algebras, that is, algebras whose unary term operations are trivial. We also study how the tractability of a finite algebra relates to the tractability of its smaller derived algebras. In Section 4 we use the results obtained to classify all strictly simple surjective algebras. We show that such algebras give rise to constraint satisfaction problems which are either tractable or **NP**-complete. Thus we extend Schaefer's dichotomy theorem to a much larger class of problems.

2 Definitions

2.1 Constraint Satisfaction Problems

The central notion in the study of constraints and constraint satisfaction problems is the notion of a *relation*. An n-ary relation on A is a subset of the set A^n, where A^n denotes the set of all n-tuples of elements of A. The set of all finitary relations on A is denoted by R_A.

We now define the standard constraint satisfaction problem which has been widely studied [1,10,11,12,14].

Definition 1. *The* constraint satisfaction problem (**CSP**) *is the combinatorial decision problem with*

Instance: *a triple* $(V, A, \mathcal{C})$ *where* V *is a set of* variables*;* A *is a domain of* values*; and* $\mathcal{C}$ *is a set of* constraints*,* $\{C_1, \ldots, C_q\}$*.*
Each constraint $C_i \in \mathcal{C}$ *is a pair* $\langle s_i, \rho_i \rangle$*, where* s_i *is a tuple of variables of length* m_i*, called the* constraint scope*, and* ρ_i *is an* m_i*-ary relation on* A*, called the* constraint relation*.*

Question: *does there exist a* solution*, i.e. a function* f*, from* V *to* A*, such that for each constraint* $\langle s_i, \rho_i \rangle \in \mathcal{C}$*, with* $s_i = (x_{i_1}, \ldots, x_{i_m})$*, the tuple* $(f(x_{i_1}), \ldots, f(x_{i_m}))$ *belongs to* ρ_i*?*

Example 1. An instance of the standard propositional SATISFIABILITY problem [4] is specified by giving a formula in propositional logic, (that is, a conjunction of clauses), and asking whether there are values for the variables which make the formula true.

Suppose that $\phi = X_1 \wedge \cdots \wedge X_n$ is such a formula, where the X_i are clauses. The satisfiability question for ϕ can be expressed as the instance $(V, \{0,1\}, \mathcal{C})$ of **CSP**, where V is the set of all variables appearing in the clauses X_i, and $\mathcal{C} = \{\langle s_1, \rho_1\rangle, \ldots, \langle s_n, \rho_n\rangle\}$. The constraints $\langle s_k, \rho_k\rangle$ are constructed as follows.

For every clause X_k, where $x_1^k, \ldots, x_{j_k}^k$ are the variables appearing in X_k, let $s_k = (x_1^k, \ldots, x_{j_k}^k)$ and $\rho_k = \{0,1\}^{j_k} \setminus \{(a_1, \ldots, a_{j_k})\}$ where $a_i = 0$ if x_i^k is negated and $a_i = 1$ otherwise (i.e., ρ_k consists of all j_k-tuples that make X_k true).

It is easy to show that solutions of this **CSP** instance are exactly assignments which make the formula ϕ true. Hence, any instance of SATISFIABILITY can be expressed as an instance of **CSP**.

Example 2. An instance of GRAPH UNREACHABILITY consists of a graph $G = (V, E)$ and a pair of vertices, $v, w \in V$. The question is whether there is no path in G from v to w. This can expressed as the **CSP** instance $(V, \{0,1\}, \mathcal{C})$ where

$$\mathcal{C} = \{\langle e, \{(0,0),(1,1)\}\rangle | e \in E\} \cup \{\langle (v), \{(0)\}\rangle, \langle (w), \{(1)\}\rangle\}.$$

A number of other examples of well-known problems expressed as **CSP**s can be found in [8].

The general constraint satisfaction problem is known to be **NP**-complete (as seen from Example 1). However, certain restrictions may affect the complexity of **CSP**. One of the possible natural ways to restrict **CSP** is to limit the relations which can appear in constraints.

Definition 2. *Let Γ be a subset of R_A. Denote by* **CSP**(Γ) *the subclass of* **CSP** *defined by the following property: any constraint relation in any instance must belong to Γ.*

Example 3. An instance of NOT-ALL-EQUAL SATISFIABILITY [4] consists of a collection of ternary clauses. The question is whether there is an assignment of values to the variables such that the variables in each clause do not all receive the same value.

This problem can be expressed as the problem **CSP**$(\{N\})$ where N is the following ternary relation on $\{0,1\}$:

$$N = \{(a,b,c) \mid \{a,b,c\} = \{0,1\}\}.$$

Example 4. An instance of GRAPH q-COLOURABILITY consists of a graph G. The question is whether the vertices of G can be labelled with q colours so that adjacent vertices are assigned different colours.

This problem can be expressed as the problem $\mathbf{CSP}(\{\neq_B\})$ where B is a q-element set (of colours) and $\neq_B$ is the disequality relation on B, defined by

$$\neq_B \ = \{(a,b) \in B^2 \mid a \neq b\}.$$

It is well known (see [4]) that this problem belongs to $\mathbf{P}$ if $q \leq 2$, and is $\mathbf{NP}$-complete otherwise.

For $A = \{0,1\}$, the time complexity of $\mathbf{CSP}(\Gamma)$ was completely classified by Schaefer [17]. In particular, he proved that such a problem is either tractable (i.e., solvable in polynomial time) or $\mathbf{NP}$-complete. The classification problem for larger domains is still open and seems to be very interesting and hard [3].

Problem 1. Characterise all sets of relations Γ such that $\mathbf{CSP}(\Gamma)$ is tractable.

2.2 Operations and Clones

By an n-ary operation on a set A we mean any mapping $f : A^n \to A$. The set of all finitary operations on A is denoted by O_A. An operation f which satisfies an identity of the form $f(x_1, \ldots, x_n) = x_i$ is called a *projection.*

Definition 3 ([18]). *A set $C \subseteq O_A$ is called a* clone *on A if*

- *C contains all projections; and*
- *C is closed under composition.*

Suppose $f(x_1, \ldots, x_n) \in O_A$ and let M be an $m \times n$ matrix whose entries a_{ij} are elements from A. Define

$$f(M) = \begin{pmatrix} f(a_{11}, \ldots, a_{1n}) \\ \vdots \\ f(a_{m1}, \ldots, a_{mn}) \end{pmatrix}.$$

Definition 4 ([13,18]). *An n-ary operation $f \in O_A$* preserves *an m-ary relation $\rho \in R_A$ (or f is a* polymorphism *of ρ, or ρ is* invariant *under f) if, for any $m \times n$ matrix M whose columns belong to ρ, the column $f(M)$ belongs to ρ as well.*

For given $F \subseteq O_A$ and $\Gamma \subseteq R_A$, let

$$Inv(F) = \{\rho \in R_A \mid \rho \text{ is invariant under each operation from } F\};$$
$$Pol(\Gamma) = \{f \in O_A \mid f \text{ is a polymorphism of each relation from } \Gamma\}.$$

Example 5. Let $A = \{0, 1, \ldots, p-1\}$, for some prime p, and let f be the ternary operation on A given by $f(x,y,z) = x - y + z \bmod p$. The set $Inv(\{f\})$ consists of all relations which are solutions to some set of linear equations modulo p. Hence, $\mathbf{CSP}(Inv(\{f\}))$ is tractable.

Jeavons has proved a strong theorem connecting the complexity of $\mathbf{CSP}(\Gamma)$ to the clone of operations $Pol(\Gamma)$.

Theorem 1 ([9]). *Let A be a finite set, and $\Gamma, \Gamma_0 \subseteq R_A$. If Γ_0 is finite and $Pol(\Gamma) \subseteq Pol(\Gamma_0)$ then $\mathbf{CSP}(\Gamma_0)$ is reducible in polynomial time to $\mathbf{CSP}(\Gamma)$.*

Informally speaking, Theorem 1 says that the complexity of $\mathbf{CSP}(\Gamma)$ is determined by the clone $Pol(\Gamma)$. Hence it allows us to tackle problems concerning the complexity of constraint satisfaction problems using algebraic techniques.

Example 6. When $A = \{0, 1\}$ all possible clones were fully described by Post [16]. Using Theorem 1, together with this description of the possible clones, it is possible to classify the complexity of any set of Boolean relations [9]. It turns out that in this case there are just 6 maximal tractable sets of relations corresponding to 6 distinct clones. For any set of relations Γ which is not contained in one of these 6 maximal sets, the corresponding problem $\mathbf{CSP}(\Gamma)$ is **NP**-complete.

This algebraic approach therefore provides an alternative proof of Schaefer's well-known dichotomy result [17] concerning the complexity of the Generalised Satisfiability problem (for details see [9]).

2.3 Algebras

In this subsection we first give the basic definitions of an algebra and certain standard algebraic constructions (following [13] and [18]). We then introduce the notion of a "tractable algebra", and show how this relates to the tractability of restricted constraint satisfaction problems.

Definition 5. *An* algebra *is an ordered pair $\mathcal{A} = (A, F)$ such that A is a nonempty set and F is a family of finitary operations on A. The set A is called the* universe *of $\mathcal{A}$; the operations from F are called* basic*. An algebra with a finite universe is referred to as a* finite algebra*.*

Definition 6. *Let $\mathcal{A} = (A, F)$ be an algebra. The m-*th direct power *of $\mathcal{A}$ is the algebra $\mathcal{A}^m = (A^m, F)$ where we treat each operation $f_i \in F$ as acting on A^m component-wise, i.e.*

$$f_i((a_{11}, \ldots, a_{1m}), \ldots, (a_{n_i 1}, \ldots, a_{n_i m})) = \\ (f_i(a_{11}, \ldots, a_{n_i 1}), \ldots, f_i(a_{1m}, \ldots, a_{n_i m})) \ .$$

Definition 7. *Let $\mathcal{A} = (A, F)$ be an algebra and B a subset of A such that, for any $f_i \in F$, and for any $b_1, \ldots, b_{n_i} \in B$, we have $f_i(b_1, \ldots, b_{n_i}) \in B$. Then the algebra $\mathcal{B} = (B, F|_B)$, where $F|_B$ consists of the restrictions of the operations f_i to B, is called a* subalgebra *of $\mathcal{A}$.*

Definition 8. *The* finite power subuniverse system *of an algebra $\mathcal{A}$ is the set $SP_{\text{fin}}(\mathcal{A})$, where*

$$SP_{\text{fin}}(\mathcal{A}) = \{\rho \mid \rho \text{ is a universe of a subalgebra of } \mathcal{A}^m \text{ for some } m \geq 1\}.$$

Remark 1. $SP_{\text{fin}}(\mathcal{A}) = Inv(F) \subseteq R_A$.

Definition 9. *Let $\mathcal{A} = (A, F)$ be an algebra. The least clone on A containing F is called the* clone of term operations *of $\mathcal{A}$ and is denoted by $Clo(\mathcal{A})$. Two algebras with the same universe are called* term equivalent *if they have the same clone of term operations.*

In other words, the clone $Clo(\mathcal{A})$ consists of all operations that can be obtained from F and the projections by means of composition.

Theorem 2 ([18]). *For a finite algebra $\mathcal{A}$, $SP_{\text{fin}}(\mathcal{A})$ is the greatest set Γ with $Pol(\Gamma) = Clo(\mathcal{A})$.*

Corollary 1. *If finite algebras $\mathcal{A}_1$ and $\mathcal{A}_2$ are term equivalent then the equality $SP_{\text{fin}}(\mathcal{A}_1) = SP_{\text{fin}}(\mathcal{A}_2)$ holds.*

We are now in a position to link the complexity of a restricted constraint satisfaction problem $\mathbf{CSP}(\Gamma)$ to properties of a corresponding algebra. Given an arbitrary set of relations $\Gamma \subseteq R_A$, consider the algebra $\mathcal{A}_\Gamma = (A, Pol(\Gamma))$. By Theorem 2, Γ is contained in $SP_{\text{fin}}(\mathcal{A}_\Gamma)$. By Theorem 1, if $\mathbf{CSP}(\Gamma)$ is tractable, then $\mathbf{CSP}(\Gamma_0)$ is tractable for any finite subset Γ_0 of $SP_{\text{fin}}(\mathcal{A}_\Gamma)$. Hence to identify sets of relations for which $\mathbf{CSP}(\Gamma)$ is tractable, it suffices to identify when the corresponding algebra, $\mathcal{A}_\Gamma$ is tractable, in the following sense.

Definition 10. *We call an algebra $\mathcal{A}$* (globally) tractable *if $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}))$ is tractable, and* locally tractable[1] *if, for every finite $\Gamma_0 \subseteq SP_{\text{fin}}(\mathcal{A})$, $\mathbf{CSP}(\Gamma_0)$ is tractable.*

We call an algebra $\mathcal{A}$ **NP**-complete *if $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}))$ is* **NP***-complete.*

With this definition, our original problem of identifying all sets of relations for which $\mathbf{CSP}(\Gamma)$ is tractable has been reduced to the following:

Problem 2. Characterise all tractable finite algebras.

Schaefer's result [17] (see Example 6) provides a first step towards answering this problem, because it yields a complete classification of two-element algebras with respect to tractability.

Theorem 3. *A two-element algebra $\mathcal{A} = (\{0,1\}, F)$ is* **NP***-complete if it is term equivalent to an algebra $(\{0,1\}, G)$ where G is a permutation group on $\{0,1\}$. Otherwise $\mathcal{A}$ is tractable.*

The central idea in the remainder of this paper is to generalise this result by using the algebraic properties of an arbitrary algebra, $\mathcal{A}$, to determine the complexity of the associated constraint satisfaction problem, $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}))$.

[1] We must distinguish the notions of global tractability and local tractability because there is no guarantee that the following situation cannot happen: for any finite subset Γ_0 of some infinite set, Γ, of relations, there exists a particular polynomial algorithm solving $\mathbf{CSP}(\Gamma_0)$, but there is no uniform polynomial algorithm for solving all instances of $\mathbf{CSP}(\Gamma)$.

3 Tractability and Algebraic Constructions

In this section we first show that when studying the tractability of finite algebras we can restrict our attention to certain special classes of algebras.

Definition 11. *We call an algebra* surjective[2] *if all of its term operations are surjective.*

It is easy to verify that a finite algebra is surjective if and only if its unary term operations form a permutation group.

Definition 12 ([20]). *Let $\mathcal{A} = (A, F)$ be an algebra, and U a non-empty subset of A. The* induced term algebra *$\mathcal{A}|_U$ is defined as $(U, (Clo\mathcal{A})|_U)$ where $(Clo\mathcal{A})|_U = \{g|_U : g \in Clo\mathcal{A}$ and g preserves $U\}$.*

It is easy to check that $Clo(\mathcal{A}|_U) = (Clo\mathcal{A})|_U$.

Theorem 4. *For any finite algebra $\mathcal{A}$ there exists a set U such that*
i) $\mathcal{A}|_U$ is surjective;
ii) If $\mathcal{A}$ is locally tractable then so is $\mathcal{A}|_U$;
iii) If $\mathcal{A}|_U$ is (locally) tractable then so is $\mathcal{A}$.

Proof. Let $\mathcal{A} = (A, F)$ be a finite algebra, and let U be a minimal subset of A which is the range of some unary operation $f \in Clo\mathcal{A}$.

Clearly, $\mathcal{A}|_U$ is surjective by the choice of U. We now prove that it satisfies properties (ii) and (iii).

(ii) Let Γ be a finite subset of $SP_{\text{fin}}(\mathcal{A}|_U)$. We will show that $\mathbf{CSP}(\Gamma)$ is linear-time reducible to $\mathbf{CSP}(\Gamma')$ where Γ' is a suitable finite subset of $SP_{\text{fin}}(\mathcal{A})$.

Let $\mathcal{P} = (V, U, \mathcal{C})$ be an instance of $\mathbf{CSP}(\Gamma)$. For any relation $\rho \in \Gamma$, with arity m, we denote by ρ' the universe of the subalgebra of $\mathcal{A}^m$ generated by ρ, i.e.,

$$\rho' = \{g(\overline{a_1}, \ldots, \overline{a_k}) \mid k > 0,\ g \in Clo(\mathcal{A}) \text{ and } \overline{a_1}, \ldots, \overline{a_k} \in \rho\}.$$

Let $\Gamma' = \{\rho' \mid \rho \in \Gamma\}$ and set $\mathcal{P}' = (V, A, \mathcal{C}')$ where $\mathcal{C}' = \{\langle s, \rho' \rangle \mid \langle s, \rho \rangle \in \mathcal{C}\}$. Now $\mathcal{P}'$ is a problem instance of $\mathbf{CSP}(\Gamma')$ which can be obtained from $\mathcal{P}$ in linear time. Since all the projections belong to $Clo(\mathcal{A})$, we have $\rho \subseteq \rho'$, so any solution of $\mathcal{P}$ is also a solution of $\mathcal{P}'$. Conversely, suppose ψ is a solution of $\mathcal{P}'$. We shall prove that $f\psi$ is a solution of $\mathcal{P}$. For this, we show that $f(\rho') \subseteq \rho$. Let $\overline{a} \in \rho'$. Then $f(\overline{a}) = fg(\overline{a_1}, \ldots, \overline{a_k})$ for some $g \in Clo(\mathcal{A})$ and some $\overline{a_1}, \ldots, \overline{a_k} \in \rho$. The operation $fg(x_1, \ldots, x_k)$ preserves U and belongs to $Clo(\mathcal{A})$; therefore its restriction to U belongs to $Clo(\mathcal{A})|_U$. Since $\rho \in SP_{\text{fin}}(\mathcal{A}|_U)$, we get $f(\overline{a}) \in \rho$.

(iii) We shall show that $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}))$ can be reduced in linear time to $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}|_U))$. First note that, by the choice of U, $f|_U$ is a permutation, so by iterating as necessary we obtain a unary operation $f' \in Clo\mathcal{A}$ such that $f'(a) = a$ for all a in U.

[2] Note that in [19] an algebra is said to be surjective if all of its basic operations are surjective. Such algebras can have non-surjective term operations.

Let $\mathcal{P} = (V, A, \mathcal{C})$ be an instance of $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}))$, where $\mathcal{C} = \{\langle s_1, \rho_1\rangle, \ldots, \langle s_q, \rho_q\rangle\}$, and consider the triple $\mathcal{P}' = \langle V, U, \mathcal{C}'\rangle$, where $\mathcal{C}' = \{\langle s_1, f'(\rho_1)\rangle, \ldots, \langle s_q, f'(\rho_q)\rangle\}$ and $f'(\rho_i) = \{f'(\overline{a}) \mid \overline{a} \in \rho_i\}$.

We first prove that $\mathcal{P}'$ is an instance of $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}|_U))$. Since the range of f' is U, any relation of the form $f'(\rho_i)$ can be viewed as a relation on U. We need to show that, for $1 \leq i \leq q$, $f'(\rho_i)$ belongs to $SP_{\text{fin}}(\mathcal{A}|_U)$, i.e. for any $g(x_1, \ldots, x_m) \in Clo(\mathcal{A}|_U)$, and for any $\overline{a_1}, \ldots, \overline{a_m} \in f'(\rho_i)$, we have $\overline{a} = g(\overline{a_1}, \ldots, \overline{a_m}) \in f'(\rho_i)$. The operation g is the restriction of some $g' \in Clo\mathcal{A}$ to U such that g' preserves U. Suppose that $\overline{a_j} = f'(\overline{b_j})$ for some $\overline{b_j} \in \rho_i$, $1 \leq j \leq m$. Then

$$\overline{a} = g(\overline{a_1}, \ldots, \overline{a_m}) = g'(f'(\overline{b_1}), \ldots, f'(\overline{b_m})).$$

We have $\overline{a} \in \rho_i$, because all elements $\overline{b_j}$ belong to ρ_i, and the function $g'(f'(x_1), \ldots, f'(x_m)) \in Clo(\mathcal{A})$. All components of $\overline{a}$ belong to U, since g' preserves U. Finally, we have $\overline{a} = f'(\overline{a}) \in f'(\rho_i)$, since $f'(a) = a$ for any $a \in U$.

Since each $\rho_i \in \mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}))$, we have $f'(\rho_i) \subseteq \rho_i$; therefore any solution of $\mathcal{P}'$ is also a solution of $\mathcal{P}$. Furthermore, if ψ is a solution of $\mathcal{P}$ then $f'\psi$ is a solution of $\mathcal{P}'$. Clearly, $\mathcal{P}'$ can be obtained from $\mathcal{P}$ in linear time. Therefore tractability of $\mathcal{A}|_U$ implies that $\mathcal{A}$ is tractable. Finally, local tractability of $\mathcal{A}|_U$ implies local tractability of $\mathcal{A}$, since if all the constraint relations in $\mathcal{C}$ are taken from some finite set $\Gamma \subseteq SP_{\text{fin}}(\mathcal{A})$ then every constraint relation in $\mathcal{C}'$ belongs to a finite subset $\{f'(\rho) \mid \rho \in \Gamma\}$ of $SP_{\text{fin}}(\mathcal{A}|_U)$. □

The next theorem shows that we need only consider surjective algebras which are *idempotent*.

Definition 13. *An operation f on A is called idempotent if $f(x, \ldots, x) = x$ for any $x \in A$. The* full idempotent reduct *of an algebra $\mathcal{A} = (A, F)$ is the algebra $(A, Clo_{id}(\mathcal{A}))$ where $Clo_{id}(\mathcal{A})$ consists of all idempotent operations from $Clo(\mathcal{A})$.*

Theorem 5. *A finite surjective algebra $\mathcal{A}$ is locally tractable if and only if its full idempotent reduct is locally tractable.*

Proof. Omitted. See the full version of this paper [2].

The next two theorems connect the tractability of a finite algebra with the tractability of its subalgebras and *homomorphic images*.

Definition 14. *Let $\mathcal{A}_1 = (A_1, F_1)$ and $\mathcal{A}_2 = (A_2, F_2)$, where $F_1 = (f_i^1 \mid i \in I)$ and $F_2 = (f_i^2 \mid i \in I)$. A map $\varphi : A_1 \to A_2$ is called a* homomorphism *from $\mathcal{A}_1$ to $\mathcal{A}_2$ if $\varphi f_i^1(a_1, \ldots, a_{n_i}) = f_i^2(\varphi(a_1), \ldots, \varphi(a_{n_i}))$ holds for all $i \in I$ and all $a_1, \ldots, a_{n_i} \in A_1$. If the map φ is onto then $\mathcal{A}_2$ is said to be a* homomorphic image *of $\mathcal{A}_1$. A homomorphic image of a subalgebra of an algebra $\mathcal{A}$ is called a* factor *of $\mathcal{A}$.*

Theorem 6. *Let $\mathcal{A}$ be a finite algebra. Then*
(i) if $\mathcal{A}$ is (locally) tractable then so is every subalgebra of $\mathcal{A}$;
(ii) if $\mathcal{A}$ has an **NP***-complete subalgebra then $\mathcal{A}$ is* **NP***-complete itself.*

Proof. Let $\mathcal{A}'$ be a subalgebra of $\mathcal{A}$. It is easy to check that $SP_{\text{fin}}(\mathcal{A}') \subseteq SP_{\text{fin}}(\mathcal{A})$. Hence, $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}'))$ can be reduced to $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}))$ in constant time.

Now (i) and (ii) follow immediately from this reduction. □

Theorem 7. *If a finite algebra $\mathcal{A}$ is locally tractable then so is every homomorphic image of $\mathcal{A}$.*

Proof. Let $\mathcal{B}$ be a homomorphic image of $\mathcal{A}$ and let φ be the corresponding homomorphism. We will show that for any finite $\Gamma \subseteq SP_{\text{fin}}(\mathcal{B})$, $\mathbf{CSP}(\Gamma)$ is linear-time reducible to $\mathbf{CSP}(\Gamma')$ for some finite $\Gamma' \subseteq SP_{\text{fin}}(\mathcal{A})$.

For $\rho \in SP_{\text{fin}}(\mathcal{B})$, set $\varphi^{-1}(\rho) = \{\overline{a} \mid \varphi(\overline{a}) \in \rho\}$ where φ acts component-wise. It is clear that $\varphi^{-1}(\rho)$ is a relation of the same arity as ρ. It is straightforward to check that $\varphi^{-1}(\rho) \in SP_{\text{fin}}(\mathcal{A})$. Let $\Gamma' = \{\varphi^{-1}(\rho) \mid \rho \in \Gamma\}$. Then Γ' is a finite subset of $SP_{\text{fin}}(\mathcal{A})$.

Take an instance $\mathcal{P} = (V, B, \mathcal{C})$ of $\mathbf{CSP}(\Gamma)$ and construct the instance $\mathcal{P}' = (V, A, \mathcal{C}')$ of $\mathbf{CSP}(\Gamma')$ where $\mathcal{C}' = \{\langle s, \varphi^{-1}(\rho)\rangle \mid \langle s, \rho\rangle \in \mathcal{C}\}$.

If ψ is a solution of $\mathcal{P}'$ then $\varphi\psi$ is a solution of $\mathcal{P}$. Conversely, if ξ is a solution of $\mathcal{P}$, then any function $\psi : V \to A$ such that $\varphi\psi(v) = \xi(v)$ for any $v \in V$ is a solution of $\mathcal{P}'$. □

Corollary 2. *If $\mathcal{A}$ is a locally tractable finite algebra then so is every factor of $\mathcal{A}$.*

The following corollary from Theorems 6 and 7 seems to be quite significant, since all known forms of **NP**-complete **CSP**s (see [15,17]) can be obtained using it. An algebra (A, G) where G is a permutation group on A is said to be a *G-set*. A G-set is said to be *non-trivial* if its universe is not a singleton.

Corollary 3. *A finite algebra $\mathcal{A}$ is* **NP***-complete if it has a factor which is term equivalent to a non-trivial G-set.*

Proof. Denote by $\mathcal{A}'$ a subalgebra of $\mathcal{A}$ such that $\mathcal{B}$ is a homomorphic image of $\mathcal{A}'$, and let B be the universe of $\mathcal{B}$.

First, consider the case when B contains only two elements, say 0 and 1. Then the relation N of Example 3 belongs to $SP_{\text{fin}}(\mathcal{B}) = Inv(G)$. It follows from the proof of Theorem 7 that $\mathbf{CSP}(\{N\})$ is reducible in polynomial time to $\mathbf{CSP}(\Gamma')$ for some finite $\Gamma' \subseteq SP_{\text{fin}}(\mathcal{A}')$. Since Not-All-Equal Satisfiability which corresponds to $\mathbf{CSP}(\{N\})$ is **NP**-complete [17], we see that $\mathcal{A}'$ is also **NP**-complete. Applying Theorem 6, we conclude that $\mathcal{A}$ is **NP**-complete.

Now consider the case when $|B| \geq 3$. Then the disequality relation $\neq_B$ of Example 4 belongs to $SP_{\text{fin}}(\mathcal{B})$. Proceeding as in the previous case one can show that Graph $|B|$-Colourability (see Example 4), which is **NP**-complete [4], is reducible in polynomial time to $\mathbf{CSP}(SP_{\text{fin}}(\mathcal{A}))$. Hence, by Theorem 6, $\mathcal{A}$ is **NP**-complete in this case also. □

4 Strictly Simple Surjective Algebras

Using the results of the previous section, we now consider certain special algebras all of whose smaller factors are tractable.

Definition 15. *A finite algebra is called* strictly simple[3] *if all of its smaller factors contain one element.*

All finite strictly simple surjective algebras have been described by Szendrei [19]. Using this powerful algebraic result we are able to classify the complexity of all such algebras.

Theorem 8. *A finite strictly simple surjective algebra $\mathcal{A}$ is* **NP***-complete if it is term equivalent to a non-trivial G-set. Otherwise $\mathcal{A}$ is tractable.*

Proof. Omitted. See the full version of the paper [2].

Example 7. Let $A = \{0, 1, \ldots, m\}$, choose $k > 2$, and let $\Gamma_k = \{R_0, R_k, \{(0)\}, \{(1)\}, \ldots, \{(m)\}\}$, where

$$R_0 = \{(0,0), (1,2), (2,3), \ldots, (m-1, m), (m, 1)\};$$
$$R_k = \{(a_1, \ldots, a_k) \in A^k \mid a_i = 0 \text{ for at least one } i,\ 1 \leq i \leq k\}.$$

Using Szendrei's classification [19], it can be shown that $\mathcal{A}_{\Gamma_k}$ is a strictly simple surjective algebra, for every k. Furthermore, $\mathcal{A}_{\Gamma_k}$ contains essentially non-unary operations, and so is not term-equivalent to a G-set. Hence, by Theorem 8, $\mathbf{CSP}(\Gamma_k)$ is tractable.

Theorem 8 is a dichotomy result: it shows that any finite strictly simple surjective algebra is either tractable or **NP**-complete. Note that Schaefer's result for the Generalised Satisfiability problem (Theorem 3) is a special case of this result. Whether there is such a dichotomy in general is possibly the most intriguing open question in this area.

Problem 3. Is every finite algebra either tractable or **NP**-complete?

The results obtained via the algebraic approach and, in particular, the results obtained in this paper, prompt us to make the following conjecture.

Conjecture 1. A finite algebra is **NP**-complete if it has a factor which is term equivalent to a non-trivial G-set. Otherwise it is tractable.

References

1. W. Bibel. Constraint satisfaction from a deductive viewpoint. *Artificial Intelligence,* 35, 1988, 401–413.

[3] In some papers appearing before 1990 such algebras are called *plain*.

2. A. Bulatov, A. Krokhin, P. Jeavons. Constraint satisfaction problems and finite algebras. Oxford University Computing Lab. Technical Report, PRG-TR-4-99.
3. T. Feder, M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: a study through Datalog and group theory. *SIAM J. Computing,* 28 (1), 1998, 57–104.
4. M. Garey, D.S. Johnson. *Computers and Intractabilty: A Guide to the Theory of NP-Completeness.* Freeman, San-Fransisco, CA, 1979.
5. M. Gyssens, P. Jeavons, D. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artificial Intelligence,* 66 (1), 1994, 57–89.
6. P. Jeavons, M.C. Cooper. Tractable constraints on ordered domains. *Artificial Intelligence,* 79 (2), 1995, 327–339.
7. P. Jeavons, D. Cohen, M. Gyssens. Closure properties of constraints. *Journal of the ACM,* 44 (4), 1997, 527–548.
8. P. Jeavons, D. Cohen, J. Pearson. Constraints and universal algebra. *Annals of Mathematics and Artificial Intelligence* 24, 1998, 51–67.
9. P. Jeavons. On the algebraic structure of combinatorial problems. *Theoretical Computer Science,* 200, 1998, 185–204.
10. L. Kirousis. Fast parallel constraint satisfaction. *Artificial Intelligence,* 64, 1993, 147–160.
11. A.K. Mackworth. Consistency in networks of relations. *Artificial Intelligence,* 8, 1977, 99–118.
12. A.K. Mackworth. Constraint satisfaction. In *Encyclopedia of Artificial Intelligence,* vol.1 (Ed. S.C. Shapiro), Wiley Interscience, 1992, 285–293.
13. R.N. McKenzie, G.F. McNulty, W. Taylor. *Algebras, Lattices, Varieties,* vol.1, Wadsworth and Brooks, California, 1987.
14. U. Montanari. Networks of constraints: fundamental properties and applications to picture processing. *Information Sciences,* 7, 1974, 95–132.
15. J. Pearson, P. Jeavons. A survey of tractable constraint satisfaction problems. Technical Report CSD-TR-97-15, Royal Holloway, University of London, 1997.
16. E.L. Post. *The two-valued iterative systems of mathematical logic,* Ann. Math. Studies 5, Princeton Univ. Press, 1941.
17. T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of 10th ACM Symposium on Theory of Computing (STOC),* 1978, 216–226.
18. Á. Szendrei. *Clones in universal algebra* (Séminaire de Mathématiques Supérieurs, vol. 99). Les Presses de l'Université de Montréal, 1986.
19. Á. Szendrei. Simple surjective algebras having no proper subalgebras, *J. Austral. Math. Soc. (Series A),* 48, 1990, 434–454.
20. Á. Szendrei. Term minimal algebras. *Algebra Universalis,* 32 (4), 1994, 439–477.

An Optimal Online Algorithm for Bounded Space Variable-Sized Bin Packing

Steven S. Seiden

Department of Computer Science
298 Coates Hall
Louisiana State University
Baton Rouge, LA 70803
sseiden@acm.org

Abstract. An online algorithm for variable-sized bin packing, based on the HARMONIC algorithm of Lee and Lee [11], is investigated. This algorithm is based on one proposed by Csirik [4]. It is shown that the algorithm is optimal among those which use bounded space. An interesting feature of the analysis is that, although it is shown that our algorithm achieves a performance ratio arbitrarily close to the optimum value, it is not known precisely what that value is. The case where bins of capacity 1 and $\alpha \in (0, 1)$ are used is studied in greater detail. It is shown that among algorithms which are allowed to chose α, the optimal performance ratio lies in $[1.37530, 1.37532]$.

Track A Keywords: Bin packing, Online algorithms, Lower bounds.

1 Introduction

Bin packing is one of the oldest and most well studied problems in computer science [5,3]. Ideas which originated in the study of the bin packing problem have helped shape computer science as we know it today. The idea of finding the best approximation algorithm for a problem has its origins in bin packing. The study of online algorithms also has its roots in the study of bin packing. In this paper, we investigate a natural generalization of the classical bin packing problem known as variable-sized bin packing.

In the *classical bin packing* problem, we receive a sequence σ of *pieces* p_1, $p_2, \ldots, p_N$. Each piece has a fixed *size* in $(0, 1]$. In a slight abuse of notation, we use p_i to indicate both the ith piece and its size. The usage should be obvious from the context. We have an infinite number of *bins* each with *capacity* 1. Each piece must be assigned to a bin. Further, the sum of the sizes of the pieces assigned to any bin may not exceed its capacity. A bin is *empty* if no piece is assigned to it, otherwise it is *used*. The goal is to minimize the number of bins used.

The *variable-sized bin packing* problem differs from the classical one in that bins do not all have the same capacity. There are an infinite number of bins of

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 283–295, 2000.
© Springer-Verlag Berlin Heidelberg 2000

each capacity $\alpha_1 < \alpha_2 < \cdots < \alpha_m = 1$. The goal now is to minimize the sum of the capacities of the bins used.

In the *online* versions of these problems, each piece must be assigned in turn, without knowledge of the next pieces. Since it is impossible in general to produce the best possible solution when computation occurs online, we consider approximation algorithms. Basically, we want to find an algorithm which incurs a cost which is within a constant factor of the minimum possible cost, no matter what the input is. This constant factor is known as the asymptotic performance ratio.

A bin-packing algorithm uses *bounded space* if it has only a constant number of bins available to accept items at any point in time. These bins are called *open* bins. Bins which have already accepted some items, but which the algorithm no longer considers for packing are *closed* bins. The bounded space assumption is a quite natural one, especially so in online bin packing. Essentially the bounded space restriction guarantees that output of packed bins is steady, that the packer does not accumulate an enormous backlog of bins which are only output at the end of processing. In many applications this is a quite desirable feature in an algorithm.

We define the asymptotic performance ratio more precisely. For a given input sequence σ, let $\mathrm{cost}_{\mathcal{A}}(\sigma)$ be the sum of the capacities of bins used by algorithm $\mathcal{A}$ on σ. Let $\mathrm{cost}(\sigma)$ be the minimum possible cost to pack pieces in σ. The *asymptotic performance ratio* for an algorithm $\mathcal{A}$ is defined to be

$$R_{\mathcal{A}}^{\infty} = \lim_{n\to\infty} \sup \left\{ \frac{\mathrm{cost}_{\mathcal{A}}(\sigma)}{\mathrm{cost}(\sigma)} \middle| \mathrm{cost}(\sigma) = n \right\}.$$

The *optimal asymptotic performance* ratio is defined to be

$$R_{\mathrm{OPT}}^{\infty} = \inf_{\mathcal{A}} R_{\mathcal{A}}^{\infty}.$$

Our goal is to find an algorithm with asymptotic performance ratio close to $R_{\mathrm{OPT}}^{\infty}$.

We now briefly review what is known about classical and variable-sized online bin-packing.

The classical online bin packing problem was first investigated by Johnson [7,8]. He showed that the Next Fit algorithm has performance ratio 2. Subsequently, it was shown by Johnson, Demers, Ullman, Garey and Graham that the First Fit algorithm has performance ratio $\frac{17}{10}$ [9]. Yao showed that Revised First Fit has performance ratio $\frac{5}{3}$, and further showed that no online algorithm has performance ratio less than $\frac{3}{2}$ [16]. Brown and Liang independently improved this lower bound to 1.53635 [1,12]. Define

$$u_{i+1} = u_i(u_i - 1) + 1, \qquad u_1 = 2,$$

and

$$h_\infty = \sum_{i=1}^{\infty} \frac{1}{u_i - 1} \approx 1.69103.$$

Lee and Lee showed that the HARMONIC algorithm, which uses bounded space, achieves a performance ratio arbitrarily close to h_∞ [11]. They further showed that no bounded space online algorithm achieves a performance ratio less than h_∞ [11]. In addition, they developed the REFINED HARMONIC algorithm, which they showed to have a performance ratio of $\frac{373}{228} < 1.63597$. The next improvement was MODIFIED HARMONIC, which Ramanan, Brown, Lee and Lee showed to have a performance ratio of $\frac{538}{333} < 1.61562$ [13]. The best claimed result to date is that of Richey [14]. He presents an algorithm called HARMONIC+1, and gives an argument that it has performance ratio 1.58872. In the author's opinion, Richey's argument lacks the rigor of a proof. The lower bound for online bin packing was improved to 1.5401 by van Vliet [15].

The variable-sized bin-packing problem was first investigated by Frieson and Langston [6]. Kinnerly and Langston gave an online algorithm with performance ratio $\frac{7}{4}$ [10]. Csirik proposed the VARIABLE HARMONIC algorithm, and showed that it has performance ratio at most h_∞ [4]. This algorithm is based on the HARMONIC algorithm of Lee and Lee [11]. Like HARMONIC, it uses bounded space. Csirik also showed that if the algorithm has two bin sizes 1 and $\alpha < 1$, and that if it is allowed to pick α, then a performance ratio of $\frac{7}{5}$ is possible [4]. Subsequent authors have investigated this problem [2,17], but Csirik's result yields the best performance ratio to date. No lower bounds are known.

In this work, we investigate the VARIABLE HARMONIC (VH) algorithm. We give a precise analysis of its performance ratio, and show that it is an optimal bounded space algorithm, by providing the first lower bound for variable-sized bin packing. An interesting feature of the analysis is that, although it is shown that our algorithm achieves a performance ratio arbitrarily close to the optimum value, it is not known precisely what that value is. The case where bins of capacity 1 and $\alpha \in (0,1)$ are used is studied in greater detail. It is shown that among algorithms which are allowed to chose α, the optimal performance ratio (and that of VH) lies in $[1.37530, 1.37532]$.

2 The VARIABLE HARMONIC Algorithm

Before we describe the algorithm we require a few definitions.

The algorithm uses as a subroutine the NEXT FIT algorithm [7,8]. This online algorithm maintains a single *open* bin. If the current item fits into the open bin, it is placed there. Otherwise, the open bin is *closed* and a new open bin is allocated. Obviously, this algorithm is online, runs in linear time and uses constant space.

VH operates by classifying pieces according to a set of predefined intervals. The algorithm has a parameter $\epsilon \in (0,1]$. $\mathbb{N} = \{0,1,2,\ldots\}$ is the set of natural numbers and $\mathbb{N}^+ = \mathbb{N} - \{0\}$. We define

$$T_i = \left\{ \frac{\alpha_i}{j} \middle| j \in \mathbb{N}^+, \alpha_i > j\epsilon \right\}, \qquad T = \bigcup_{i=1}^{m} T_i.$$

Let the members of T be $t_1 > t_2 > \cdots > t_n$. We define $t_{n+1} = \epsilon$ and $t_{n+2} = 0$. The interval I_j is defined to be $(t_{i+1}, t_i]$ for $j = 1, \ldots, n+1$. Note that these intervals are disjoint and that they cover $(0, 1]$.

A piece of size s has *type* j if $s \in I_j$. The *class* and *order* of interval I_k are i and j, respectively, if $t_k = \alpha_i/j$ (breaking ties arbitrarily). We extend the definitions of class and order to include pieces in the natural way. A piece is *big* if it has type $i \leq n$.

The algorithm packs pieces of different types independently. I.e. pieces of differing types never appear in the same bin. Pieces of type $n+1$ are packed using Next Fit into bins of capacity 1. Pieces of class i and order j are packed j to a bin in bins of capacity α_i. The algorithm keeps one open bin for each type, into which pieces are packed until the indicated number is reached, at which point it is closed and a new open bin is allocated. Since the number of types (and thus the number of open bins) is constant, the algorithm is online, runs in linear time and uses constant space. Note that when $m = 1$ the definition of VH corresponds exactly with that of Harmonic.

3 An Upper Bound for VH

The analysis is based on *weighting functions* as in [11,4]. A weighting function for algorithm $\mathcal{A}$ is a function $w_{\mathcal{A}} : (0, 1] \mapsto [0, 1]$ with the property

$$\mathrm{cost}_{\mathcal{A}}(\sigma) \leq \sum_{i=1}^{N} w_{\mathcal{A}}(p_i) + O(1),$$

for all input sequences $p_1, \ldots, p_N$. Intuitively, the weight of a piece indicates the maximum portion of a bin that it can occupy. We use the following function:

$$w_{\mathrm{VH}}(x) = \begin{cases} t_i & \text{if } x \in I_i \text{ with } i \leq n, \\ \dfrac{1}{1-\epsilon} x & \text{if } x \in I_{n+1}. \end{cases}$$

Lemma 1. *For all σ,*

$$\mathrm{cost}_{\mathrm{VH}}(\sigma) \leq \sum_{i=1}^{N} w_{\mathrm{VH}}(p_i) + n + 1.$$

Proof. We consider first the cost of bins used to pack pieces of an arbitrary individual type. Next Fit is used to pack type $n+1$ pieces. Each of these pieces is of size ϵ or less. Therefore, each of these bins is filled to within ϵ of its capacity. Let X be the total size of type $n+1$ pieces. The number of bins used is at most

$$\left\lceil \frac{1}{1-\epsilon} X \right\rceil \leq \frac{1}{1-\epsilon} X + 1 = \frac{1}{1-\epsilon} \sum_{p_i \in I_{n+1}} p_i + 1 = \sum_{p_i \in I_{n+1}} w_{\mathrm{VH}}(p_i) + 1.$$

Consider now pieces of type $i \leq n$. Let j and k be the class and order of I_i, respectively. These pieces are packed k to a bin in bins of capacity α_j. Let ℓ be the number of them. Then the number of bins used is $\lceil \ell/k \rceil$. The cost to the algorithm is

$$\alpha_j \left\lceil \frac{\ell}{k} \right\rceil \leq \frac{\alpha_j \ell}{k} + 1 = t_i \ell + 1 = \sum_{p_i \in I_i} w_{\mathrm{VH}}(p_i) + 1.$$

Summing over all types gives the desired result. □

Now consider the optimal packing. Let n_i be the number of bins of capacity α_i in the optimal packing. Let $B_{i,j}$ be the set of pieces in the jth bin of size α_i.

Suppose bin $B_{i,j}$ in the optimal packing is not full. Let $x = \sum_{p \in B_{i,j}} p$. Then add a piece of size $\alpha_i - x$ to the end of our sequence. The cost of the optimal solution does not increase, whereas the cost to the online algorithm cannot decrease. Therefore, when upper bounding the performance ratio of an online algorithm, we may assume that each bin in the optimal packing is full.

¿From the preceding definitions we have

$$\mathrm{cost}(\sigma) = \sum_{i=1}^{m} \alpha_i \cdot n_i.$$

To show that the algorithm has performance ratio at most c, we show that

$$\sum_{i=1}^{N} w_{\mathrm{VH}}(p_i) \Big/ \sum_{i=1}^{m} \alpha_i \cdot n_i \leq c,$$

for all σ.

We find

$$\begin{aligned} \sum_{i=1}^{N} w_{\mathrm{VH}}(p_i) \Big/ \sum_{i=1}^{m} \alpha_i \cdot n_i &= \sum_{i=1}^{m} \sum_{j=1}^{n_i} \sum_{p \in B_{i,j}} w_{\mathrm{VH}}(p) \Big/ \sum_{i=1}^{m} \alpha_i \cdot n_i \\ &= \sum_{i=1}^{m} \sum_{j=1}^{n_i} \sum_{p \in B_{i,j}} w_{\mathrm{VH}}(p) \Big/ \sum_{i=1}^{m} \sum_{j=1}^{n_i} \alpha_i. \end{aligned}$$

For the above value to be greater than c, for some i and j we must have

$$\sum_{p \in B_{i,j}} w_{\mathrm{VH}}(p) > c \cdot \alpha_i.$$

If we show that this is impossible, we show that the algorithm has performance ratio at most c.

We are therefore led to consider the following optimization problem: Maximize

$$\sum_{p \in X} w_{\mathrm{VH}}(p)/\alpha_i,$$

subject to $\sum_{p \in X} p = \alpha_i$ and $i \in \{1, \ldots, m\}$. We further rewrite this as: Maximize

$$\frac{1}{\alpha_i}\left(\sum_{j=1}^{n} q_j \cdot t_j + \frac{\alpha_i - y}{1-\epsilon}\right), \tag{1}$$

subject to

$$y < \alpha_i, \tag{2}$$
$$q_j \in \mathbb{N}, \qquad 1 \le j \le n, \tag{3}$$
$$i \in \{1, \ldots, m\}, \tag{4}$$

where we define

$$y = \sum_{j=1}^{n} q_j \cdot t_{j+1}. \tag{5}$$

Intuitively, q_j is the number of type j pieces. y is a lower bound on the sizes of the big pieces. Note that strict inequality is required in (2) because a type j piece is strictly larger than t_{j+1}. We shall consider this mathematical program to be a function of ϵ, which we denote as $\mathcal{P}_1(\epsilon)$.

We have shown the following:

Theorem 1. *The performance ratio of* VH *is upper bounded by the value of* $\mathcal{P}_1(\epsilon)$.

We shall return to the evaluation of $\mathcal{P}_1(\epsilon)$ at a point later in our exposition.

4 A Lower Bound for Bounded-Space Algorithms

Consider the mathematical program: Maximize

$$\frac{1}{\alpha_i}\left(\sum_{j=1}^{n} q_j \cdot t_j + \alpha_i - y\right), \tag{6}$$

subject to (2-5). We denote this program as $\mathcal{P}_2(\epsilon)$. We have the following result:

Lemma 2. *Any feasible solution to* $\mathcal{P}_2(\epsilon)$ *with objective value* c *yields a lower bound of* c *for all bounded space variable-sized bin packing algorithms.*

Proof. Let $q_1, \ldots, q_n, i$ be a feasible solution and let c be the corresponding value of (6). Define $Q = \sum_{j=1}^{n} q_j + 1$. We create an input with $N \cdot Q$ pieces. Let $\delta > 0$ be a real number. The input consists of $n+1$ groups of pieces. All pieces in a group are the same size. The jth group contains $q_j N$ pieces of size $t_{j+1} + \delta$ for $1 \le j \le n$. The last group contains N pieces of size $z = \alpha_i - y - (Q-1)\delta$. We require that δ be chosen such that $z \ge 0$ and $t_{j+1} + \delta \in I_j$ for $1 \le j \le n$.

The optimal solution uses N bins of size α_i. Each of these bins contains q_j items from group j for $1 \le j \le n$ and one item from the last group.

Consider the number of bins used by an arbitrary bounded space online algorithm $\mathcal{A}$ on this input. Since $\mathcal{A}$ is online and uses bounded space, at most a constant number (with respect to N) of pieces in group j can be packed with pieces from other groups. Therefore, the cost to pack items in the last group is at least $Nz - O(1)$. Further, the minimum cost to pack the items in group $j \le n$ is

$$\min_{1\le k\le m} \frac{\alpha_k\, q_j N}{\lfloor \alpha_k/(t_{j+1}+\delta)\rfloor} - O(1) = q_j N \min_{1\le k\le m} \frac{\alpha_k}{\lfloor \alpha_k/(t_{j+1}+\delta)\rfloor} - O(1).$$

We assert that

$$\min_{1\le k\le m} \frac{\alpha_k}{\lfloor \alpha_k/(t_{j+1}+\delta)\rfloor} \ge t_j. \tag{7}$$

Given this fact, the asymptotic performance ratio of $\mathcal{A}$ is lower bounded by

$$\lim_{N\to\infty} \frac{N\left(\sum_{j=1}^{n} t_j \cdot q_j + \alpha_i - y - (Q-1)\delta\right) - O(1)}{\alpha_i N} = c - (Q-1)\delta,$$

which can be made arbitrarily close to c by choosing sufficiently small δ. Therefore, to complete the proof we need merely show (7).

Suppose for a contradiction that (7) is false. Then there exists a k such that

$$\frac{\alpha_k}{\lfloor \alpha_k/(t_{j+1}+\delta)\rfloor} < t_j.$$

Let ℓ be the integer $\lfloor \alpha_k/(t_{j+1}+\delta)\rfloor$. Note that since

$$\frac{(t_{j+1}+\delta)\ell}{\alpha_k} = \frac{t_{j+1}+\delta}{\alpha_k}\left\lfloor \frac{\alpha_k}{t_{j+1}+\delta}\right\rfloor \le 1,$$

we have $\alpha_k/\ell \ge t_{j+1}+\delta$. Therefore we have $\alpha_k/\ell \in [t_{j+1}+\delta, t_j) \subset I_j$. However, by the definition of I_j, we have no number of the form α_k/ℓ, ℓ an integer, within I_j. So we have reached a contradiction. □

Lemma 3.

$$\lim_{\epsilon\to 0} [\text{value}(\mathcal{P}_1(\epsilon)) - \text{value}(\mathcal{P}_2(\epsilon))] = 0.$$

Proof. Since $\mathcal{P}_1(\epsilon)$ and $\mathcal{P}_2(\epsilon)$ share the same constraints, any feasible solution to one is a feasible solution to the other. Further, for any feasible solution, the value of (1) is greater than that of (6) by

$$\frac{1}{\alpha_i}\left(\frac{\alpha_i - y}{1-\epsilon} - (\alpha_i - y)\right) = \frac{\epsilon}{1-\epsilon}\left(1 - \frac{y}{\alpha_i}\right) \le \frac{\epsilon}{1-\epsilon}.$$

Consider the assignment to $q_1, \ldots, q_n, i$ which maximizes (1). The value of (6) at this feasible solution lower bounds the value of $\mathcal{P}_2(\epsilon)$. So we have

$$\lim_{\epsilon \to 0} [\text{value}(\mathcal{P}_1(\epsilon)) - \text{value}(\mathcal{P}_2(\epsilon))] \leq \lim_{\epsilon \to 0} \frac{\epsilon}{1-\epsilon} = 0.$$

□

We are now ready to state the main result:

Theorem 2. VH *is an optimal bounded space online algorithm.*

Proof. This follows directly from Theorem 1 and Lemmas 2 and 3. □

5 Evaluating $\mathcal{P}_1$ and $\mathcal{P}_2$

We present an algorithm to evaluate $\mathcal{P}_1(\epsilon)$ and $\mathcal{P}_2(\epsilon)$. Define y as in (5). Further define:

$$\begin{aligned} e_j &= \frac{t_j}{t_{j+1}}, \qquad \text{for } 1 \leq j \leq n, \\ e_{n+1} &= \frac{1}{1-\epsilon}, \\ E_j &= \max_{j \leq k \leq n+1} e_k \qquad \text{for } 1 \leq j \leq n+1. \end{aligned}$$

The algorithm to compute $\mathcal{P}_1(\epsilon)$ is displayed in Figures 1 and 2. By simply

```
x ← 1.
For i ∈ {1, ..., m} do:
    Initialize q_j ← 0 for 1 ≤ j ≤ n.
    TRYALL(1).
Return x.
```

Fig. 1. The algorithm for computing $\mathcal{P}_1(\epsilon)$.

redefining the value of e_{n+1} to be 1, we compute instead $\mathcal{P}_2(\epsilon)$.

The main routine of the algorithm initializes variables, and iterates over the possible values of i. The variable x stores maximum objective value found at any point in the computation.

The real work of the algorithm is done in the subroutine TRYALL. This subroutine traverses an implicit data structure which we call the *feasible solution tree*. This is a rooted tree with $n+1$ levels. The edges of the tree are labeled with natural numbers. All leaves are at level $n+1$. Along any path from the root to a leaf, the label of the jth edge represents the value of q_j. Each such path specifies a feasible solution. TRYALL recursively traverses the feasible solution tree, avoiding sub-trees which cannot improve on the best solution found so far.

$z \leftarrow \frac{1}{\alpha_i}\left(\sum_{k=1}^{j} q_k \cdot t_k + (\alpha_i - y)E_j\right)$.
If $z \leq x$ then:
    Return.
Else if $j = n+1$ then:
    $x \leftarrow z$.
Else:
    $q_j \leftarrow \lceil (\alpha_i - y)/t_{j+1} \rceil$.
    While $q_j > 0$ do:
        $q_j \leftarrow q_j - 1$.
        TRYALL$(j+1)$.

Fig. 2. The subroutine TRYALL(j).

During the execution of TRYALL, j represents our current level in the tree. If $j = n+1$, then we have reached a leaf. In this case, the value z assigned in the first step of the algorithm is exactly the objective value. When $j \leq n$, the value of z is an upper bound on any objective value in the current sub-tree. If this value does not exceed x then we do not explore this sub-tree. Otherwise, we recurse for each of the possible values of q_j, from the largest down to zero. This heuristic drastically decreases the running time of the algorithm, as the first solution found is the greedy one, where by greedy we mean the largest possible item is added to the solution at each step. In fact there are several reasonable alternative defintions of greedy for the variable-size problem. The optimal solution is quite often greedy, however, there are some cases where the optimal solution is not any greedy one. We shall explain this in more detail in the full version.

6 The Dual-Capacity Problem

We now focus our attention on the case where $m = 2$, which we call the *dual-capacity* problem. In order to simplify notation, we denote the size of the smaller bin as α. The size of the larger bin is 1, as before. We investigate how the optimal asymptotic performance ratio R^{∞}_{OPT} varies as a function of α. We therefore consider the values of $\mathcal{P}_1$ and $\mathcal{P}_2$ to be functions of α, as well as ϵ. As we have shown in the preceding sections, $\mathcal{P}_2(\epsilon, \alpha) \leq R^{\infty}_{\text{OPT}}(\alpha) \leq \mathcal{P}_1(\epsilon, \alpha)$.

Using the algorithm described in Section 5, we can compute a lower bound for any fixed value of α. However, what we would like to do is prove a lower bound which holds for all α. To further this goal, we study the structure of $\mathcal{P}_2(\epsilon, \alpha)$ in more detail.

Since we only need to lower bound $\mathcal{P}_2(\epsilon, \alpha)$, for the discussion that follows we fix $i = 2$. We consider only $\epsilon = 1/M$ for $M \in \mathbb{N}^+$.

Note that, as α varies, the values in T_1 change, while the values in T_2 are fixed. For certain values of α, it may happen that a point in T_1 is coincident with one in T_2. We call such a point an *interesting* point. At an interesting point, a combinatorial change occurs in the structure of $t_1, \ldots, t_n$, since two

points exchange order. We also include zero as an interesting point. It is not hard to see that the set of interesting points is:

$$\{0\} \cup \bigcup_{\ell=1}^{M} \bigcup_{j=\ell}^{M} \frac{\ell}{j}.$$

Rename the points in this set to be $0 = s_1 < s_2 < \cdots < s_L = 1$. Define $S_j = (s_{j-1}, s_j]$. We shall explain how to lower bound the value of $\mathcal{P}_2(\epsilon, \alpha)$ for $\alpha \in S_j$.

We further split each interval S_j into disjoint sub-intervals $(u_1, u_2]$, $(u_2, u_3]$, $\ldots, (u_{\ell-1}, u_\ell]$ with $u_1 = s_{j-1}$ and $u_\ell = s_j$. Suppose that $q_1, \ldots, q_n, i = 2$ is a feasible solution to $\mathcal{P}_2(\epsilon, \alpha)$ at $\alpha = u_k$. We assert that this is feasible solution for all $\alpha \in (u_{k-1}, u_k]$. To see this, note that within $(s_{j-1}, u_k]$, y is a non-decreasing linear function of α, for any fixed assignment to $q_1, \ldots, q_n, i$. Therefore, (2) remains valid throughout $(s_{j-1}, u_k] \supset (u_{k-1}, u_k]$. Furthermore, the objective (6) is also a linear function of α for any fixed assignment to $q_1, \ldots, q_n, i$.

To get a lower bound for $(u_{k-1}, u_k]$, we evaluate $\mathcal{P}_2(\epsilon, u_k)$ (fixing $i = 2$). We record the assignment $q_1, \ldots, q_n$ which gives the highest lower bound. Substituting these values into (6), we get a linear function. This is minimized at either $\alpha = u_{k-1}$ or $\alpha = u_k$. We evaluate the function at these two points to get the desired lower bound on $(u_{k-1}, u_k]$. Iterating over all intervals and sub-intervals, we can compute a lower bound for all α.

As an example, let $\epsilon = \frac{1}{10}$. This yields 33 interesting points and 32 intervals. For simplicity's sake, we do not divide an interval into subintervals. We find that

We get a lower bound of $\frac{273}{200} = 1.365$ at $\alpha = \frac{7}{10}$. We have combined adjacent intervals where possible. For example, we find that the lower bound function in both $(\frac{3}{4}, \frac{7}{9}]$ and $(\frac{7}{9}, \frac{4}{5}]$ is $\frac{53}{60} + \frac{2}{3}\alpha$, and so we have one entry for $(\frac{3}{4}, \frac{4}{5}]$.

The main result of this section is:

Theorem 3.

$$1.37532 > \frac{395101163}{287280000} \geq \inf_{\alpha \in (0,1]} R^{\infty}_{\mathrm{OPT}}(\alpha) \geq \frac{78392621}{57000000} > 1.37530.$$

Proof. The value of $\mathcal{P}_1(1/100, 57143/80000)$ is $395101163/287280000$. ¿From Figure 3, we find that if $\mathcal{P}_2(\epsilon, \alpha) < 78392621/57000000$ then we must have $\alpha \in (7/10, 3/4]$. We determine the intervals S_j for $\epsilon = 1/100$, and eliminate those which do not overlap $(7/10, 3/4]$. This leaves 154 intervals to be checked. These intervals were divided further into sub-intervals, by including all points $7/10 + j/100000$ for $1 \leq j < 2500$. We have verified using `Mathematica` that the minimum lower bound over this set of sub-intervals is $78392621/57000000$. □

To get a better picture of the curve $R^{\infty}_{\mathrm{OPT}}(\alpha)$, we have computed values of $\mathcal{P}_1(\frac{1}{100}, \alpha)$ and $\mathcal{P}_2(\frac{1}{100}, \alpha)$ for $\alpha = j/10000, 1 \leq j \leq 10000$, using `Mathematica`. The maximum difference between the two values at any of these points was less than 0.00014. Since the difference is so small, we display just $\mathcal{P}_1$ in Figure 4.

Low α	High α	Function
0	$\frac{1}{7}$	$\frac{71}{42}$
$\frac{1}{7}$	$\frac{1}{6}$	$\frac{32}{21} + \alpha$
$\frac{1}{6}$	$\frac{2}{7}$	$\frac{71}{42}$
$\frac{2}{7}$	$\frac{3}{10}$	$\frac{283}{168}$
$\frac{3}{10}$	$\frac{1}{3}$	$\frac{32}{21} + \frac{\alpha}{2}$
$\frac{1}{3}$	$\frac{3}{8}$	$\frac{17}{9} - \frac{2\,\alpha}{3}$
$\frac{3}{8}$	$\frac{2}{5}$	$\frac{49}{30}$
$\frac{2}{5}$	$\frac{3}{7}$	$\frac{25}{21} + \alpha$
$\frac{3}{7}$	$\frac{4}{9}$	$\frac{4}{3} + \frac{2\,\alpha}{3}$
$\frac{4}{9}$	$\frac{1}{2}$	$\frac{43}{42} + \frac{4\,\alpha}{3}$
$\frac{1}{2}$	$\frac{5}{9}$	$\frac{31}{15} - \frac{4\,\alpha}{5}$
$\frac{5}{9}$	$\frac{5}{8}$	$\frac{13}{6} - \alpha$
$\frac{5}{8}$	$\frac{2}{3}$	$\frac{7}{4} - \frac{\alpha}{2}$
$\frac{2}{3}$	$\frac{7}{10}$	$\frac{25}{12} - \alpha$
$\frac{7}{10}$	$\frac{5}{7}$	$\frac{7}{8} + \frac{7\,\alpha}{10}$
$\frac{5}{7}$	$\frac{3}{4}$	$\frac{8}{9} + \frac{2\,\alpha}{3}$
$\frac{3}{4}$	$\frac{4}{5}$	$\frac{53}{60} + \frac{2\,\alpha}{3}$
$\frac{4}{5}$	$\frac{5}{6}$	$\frac{1}{42} + \frac{17\,\alpha}{10}$
$\frac{5}{6}$	$\frac{6}{7}$	$\frac{4}{21} + \frac{3\,\alpha}{2}$
$\frac{6}{7}$	$\frac{9}{10}$	$\frac{1}{3} + \frac{4\,\alpha}{3}$
$\frac{9}{10}$	1	$\frac{1}{42} + \frac{5\,\alpha}{3}$

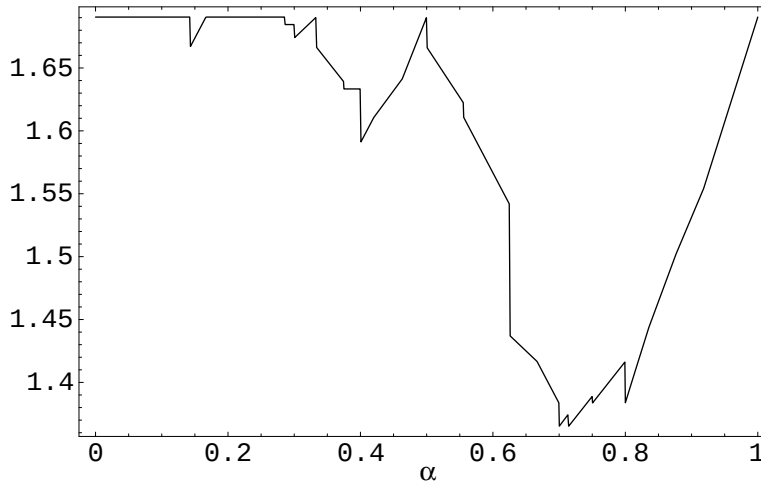


Fig. 3. The lower bound function for $\epsilon = \frac{1}{10}$.

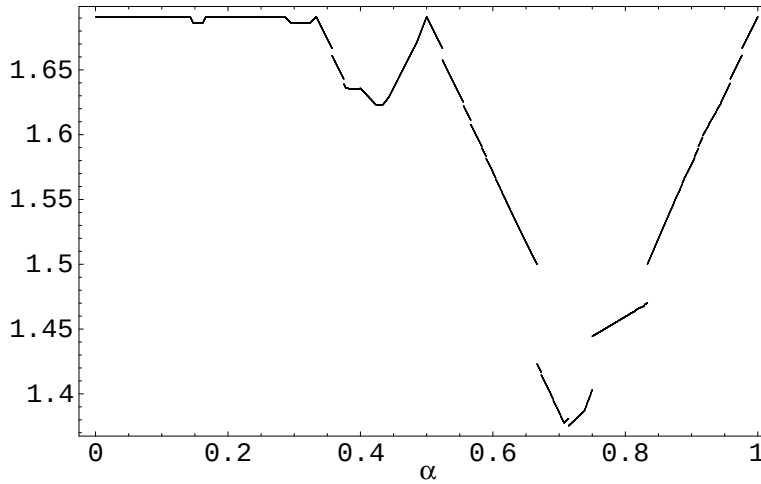


Fig. 4. Values of $\mathcal{P}_1(\frac{1}{100}, \alpha)$.

7 Conclusions

We have shown the optimality of Variable Harmonic among bounded space algorithms for variable sized bin packing. A number of open questions remain:

1. The curve $R^{\infty}_{\mathrm{OPT}}(\alpha)$ investigated in Section 6 seems to have the property that the closer we examine it, the more detail it reveals. I.e. it is "fractal like" in some sense. What can be said about this curve?
2. What general upper and lower bounds can be proved for variable-sized bin packing?

References

1. Brown, D. J. A lower bound for on-line one-dimensional bin packing algorithms. Tech. Rep. R-864, Coordinated Sci. Lab., University of Illinois at Urbana-Champaign, 1979.
2. Burkard, R., and Zhang, G. Bounded space on-line variable-sized bin packing. *Acta Cybernetica 13*, 1 (1997), 63–76.
3. Coffman, E. G., Garey, M. R., and Johnson, D. S. Approximation algorithms for bin packing: A survey. In *Approximation Algorithms for NP-hard Problems*, D. Hochbuam, Ed. PWS Publishing Company, 1997, ch. 2.
4. Csirik, J. An on-line algorithm for variable-sized bin packing. *Acta Informatica 26*, 8 (1989), 697–709.
5. Csirik, J., and Woeginger, G. On-line packing and covering problems. In *On-Line Algorithms—The State of the Art*, A. Fiat and G. Woeginger, Eds., Lecture Notes in Computer Science. Springer-Verlag, 1998, ch. 7.
6. Friesen, D. K., and Langston, M. A. Variable sized bin packing. *SIAM Journal on Computing 15* (1986), 222–230.

7. Johnson, D. S. *Near-optimal bin packing algorithms.* PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1973.
8. Johnson, D. S. Fast algorithms for bin packing. *Journal Computer Systems Science 8* (1974), 272–314.
9. Johnson, D. S., Demers, A., Ullman, J. D., Garey, M. R., and Graham, R. L. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing 3* (1974), 256–278.
10. Kinnersley, N., and Langston, M. Online variable-sized bin packing. *Discrete Applied Mathematics 22*, 2 (Feb 1989), 143–148.
11. Lee, C., and Lee, D. A simple on-line bin-packing algorithm. *Journal of the ACM 32*, 3 (Jul 1985), 562–572.
12. Liang, F. M. A lower bound for online bin packing. *Information Processing Letters 10* (1980), 76–79.
13. Ramanan, P., Brown, D., Lee, C., and Lee, D. On-line bin packing in linear time. *Journal of Algorithms 10*, 3 (Sep 1989), 305–326.
14. Richey, M. B. Improved bounds for harmonic-based bin packing algorithms. *Discrete Applied Mathematics 34* (1991), 203–227.
15. van Vliet, A. An improved lower bound for online bin packing algorithms. *Information Processing Letters 43*, 5 (Oct 1992), 277–284.
16. Yao, A. C. C. New algorithms for bin packing. *Journal of the ACM 27* (1980), 207–227.
17. Zhang, G. Worst-case analysis of the FFH algorithm for online variable-sized bin packing. *Computing 56*, 2 (1996), 165–172.

Resource Augmentation for Online Bounded Space Bin Packing
(Extended Abstract)

János Csirik[1] and Gerhard J. Woeginger[2]

[1] Department of Computer Science, University of Szeged, Hungary
[2] Institut für Mathematik, TU Graz, Austria.

Abstract. We study online bounded space bin packing in the resource augmentation model of competitive analysis. In this model, the online bounded space packing algorithm has to pack a list L of items in $(0, 1]$ into a small number of bins of size $b \geq 1$. Its performance is measured by comparing the produced packing against the optimal offline packing of the list L into bins of size 1.
We present a complete solution to this problem: For every bin size $b \geq 1$, we design online bounded space bin packing algorithms whose worst case ratio in this model comes arbitrarily close to a certain bound $\rho(b)$. Moreover, we prove that no online bounded space algorithm can perform better than $\rho(b)$ in the worst case.

Keywords. Online algorithm, competitive analysis, resource augmentation, approximation algorithm, asymptotic worst case ratio, bin packing.

1 Introduction

Resource augmentation (or *extra-resource analysis*) is a technique for analyzing online algorithms that was introduced in 1995 by Kalyanasundaram & Pruhs [4]. It is a relaxed notion of competitive analysis in which the online algorithm is given better resources than the optimal offline algorithm to which it is compared. This is e.g. the case, if the machines of the online algorithm run at slightly higher speed than those of the offline algorithm, or if the online algorithm has more machines than the offline algorithm, or if the production deadlines of the online algorithm are less stringent than those of the offline algorithm. The main idea behind the resource augmentation technique is to give the online algorithm a fairer chance in competing against the omniscient and all-powerful offline algorithm from classical competitive analysis. During the last few years the resource augmentation technique has become a very popular tool, and it has been applied to many problems in scheduling (cf. e.g. Phillips, Stein, Torng & Wein [8] and Edmonds [3]), in paging (Albers, Arora & Khanna [1]), and in combinatorial optimization (Kalyanasundaram & Pruhs [5]). In this paper we will study online bounded space bin packing in this resource augmentation model.

In the classical bin packing problem, a list $L = \langle a_1, a_2, \ldots \rangle$ of items $a_i \in [0, 1]$ has to be packed into the minimum number of unit-size bins. The *offline*

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 296–304, 2000.
© Springer-Verlag Berlin Heidelberg 2000

optimum $\textsc{Opt}_1(L)$ is the minimum number of unit-size bins into which the items in L can be fit. A bin packing algorithm is called *online* if it packs all items a_i solely on the basis of the sizes of the items a_j, $1 \leq j \leq i$, and without any information on subsequent items. A bin packing algorithm uses *k-bounded space* if for each item a_i, the choice of bins to pack it into is restricted to a set of k or fewer *active* bins. Each bin becomes active when it receives its first item, but once it is declared inactive (or *closed*), it can never become active again. An online *bounded space* bin packing algorithm is an online algorithm that uses k-bounded space for some fixed value $k \geq 1$. The bounded space restriction models situations in which bins are exported once they are packed (e.g., in packing trucks at a loading dock that has positions for only k trucks, or in communication channels with buffers of limited size in which information moves in large fixed-size blocks).

We investigate the behavior of online bounded space bin packing algorithms that pack the list L into bins of size $b \geq 1$. This larger bin size b is the augmented resource of the online algorithm; the offline algorithm has to work with bins of size 1. For an online algorithm A and a bin size b, we denote by $A_b(L)$ the number of bins of size b that algorithm A uses in packing the items in L. The *worst case performance* of algorithm A for bin size b, denoted by $R_b(A)$, is defined as

$$R_b(A) \;=\; \lim_{Opt_1(L)\to\infty} \sup_{L} \; A_b(L)/\textsc{Opt}_1(L).$$

A small worst case performance means a good quality of the online algorithm. Online bin packing is a classical problem in optimization and theoretical computer science. We refer the reader to Csirik & Woeginger [2] for an up-to-date survey of this area.

Our results and organization of the paper. In this paper we present a complete analysis of online bounded space bin packing in the resource augmentation model: For every bin size $b \geq 1$, we determine the best possible worst case performance $\rho(b)$ over all online bounded space bin packing algorithms. The precise values $\rho(b)$ are defined in Section 2. In Section 3 we state several auxiliary results. In Section 4 we discuss technical properties of the function $\rho(b)$. In Section 5 we design and analyze an online algorithm whose worst case performance comes arbitrarily close to $\rho(b)$. Finally, in Section 6 we prove that no online algorithm can beat the bound $\rho(b)$.

2 Statement of the Main Result

Throughout the paper, $L = \langle a_1, a_2, \ldots, a_n \rangle$ is a list of items in $(0, 1]$, and $b \geq 1$ is the bin size for the online algorithm. We associate with b an infinite sequence $T(b) = \langle t_1, t_2, \ldots \rangle$ of positive integers as follows:

$$t_1 = \lfloor 1 + b \rfloor \quad \text{and} \quad r_1 = \frac{1}{b} - \frac{1}{t_1}, \tag{1}$$

and for $i = 1, 2, \ldots$

$$t_{i+1} = \lfloor 1 + \frac{1}{r_i} \rfloor \quad \text{and} \quad r_{i+1} = r_i - \frac{1}{t_{i+1}}. \tag{2}$$

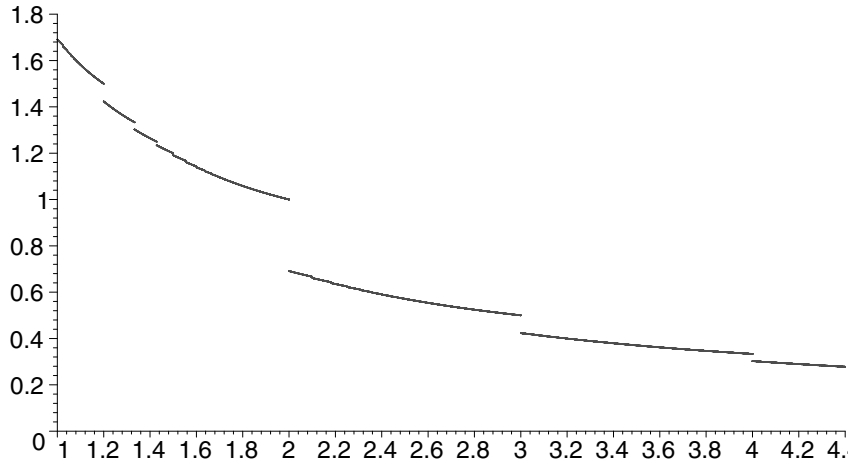


Fig. 1. The graph of the function $\rho(b)$.

An equivalent way for defining this sequence $T(b)$ is the following: Suppose that we want to fill a bucket of size $1/b$ greedily with reciprocal values of positive integers. First, we pack the largest possible reciprocal value that fits into the bucket, but without filling it completely. Then we add the largest reciprocal value that fits without filling the rest capacity completely, and then this process is repeated over and over again. In this 'bucket' interpretation, the value r_i represents the rest capacity after the reciprocal value of the positive integer t_i has been put into the bucket. Note that the smallest integer whose reciprocal would fit into a space of $r \leq 1$ is $\lceil 1/r \rceil$. If $1/r$ happens to be an integer, we must not fill the bucket completely, and hence we have to pack the reciprocal of $\lceil 1/r \rceil + 1$ instead. The reader may want to verify that the recursive definitions in (1) and (2) exactly agree with these interpretations. Alltogether, this discussion demonstrates that

$$\frac{1}{b} \;=\; \sum_{i=1}^{\infty} \frac{1}{t_i} \;=\; \frac{1}{t_1} + \frac{1}{t_2} + \frac{1}{t_3} + \frac{1}{t_4} + \cdots \tag{3}$$

Finally, we define

$$\rho(b) \;=\; \sum_{i=1}^{\infty} \frac{1}{t_i - 1}. \tag{4}$$

In Section 3 we will prove that the infinite sum in the righthand side of (4) converges for every value of b. The following lemma provides the reader with some intuition on the (somewhat irregular and somewhat messy) behaviour of the function $\rho(b)$; see also the picture in Figure 1 for an illustration. The lemma will be proved in Section 4.

Lemma 1. *The function $\rho(b) : [1, \infty) \to \mathbb{R}$ has the following properties.*

(i) $\rho(1) \approx 1.69103$ *and* $\rho(2) \approx 0.69103$.
(ii) $1/m \leq \rho(m) \leq 1/(m-1)$ *for integers* $m \geq 2$.
(iii) $\rho(b)$ *is strictly decreasing on* $[1, \infty)$.
(iv) As b *tends to* 2 *from below,* $\rho(b)$ *tends to* 1. *As* b *tends to infinity,* $\rho(b)$ *tends to* 0.
(v) At every irrational value of $b > 1$, *the function* $\rho(b)$ *is continuous.*
(vi) At every rational value of $b > 1$, *the function* $\rho(b)$ *is not continuous.*

The following theorem summarizes the main result of this paper. Its proof is split into the proof of the upper bound in Theorem 7 in Section 5, and into the proof of the lower bound in Theorem 8 in Section 6.

Theorem 2. *(Main result of the paper)*
For every bin size $b \geq 1$, *there exist online bounded space bin packing algorithms with worst case performance arbitrarily close to* $\rho(b)$. *For every bin size* $b \geq 1$, *the bound* $\rho(b)$ *cannot be beaten by an online bounded space bin packing algorithm.*

Note that by setting $b = 1$ in Theorem 2 we get a worst case performance of $\rho(1) \approx 1.69103$. Hence, this special case reproves the well-known result of Lee & Lee [6] on classical online bounded space bin packing.

3 Some Useful Facts

In this section we collect several facts on the sequence $T(b)$ that will be used in the later sections. First, we observe that for every $b \geq 1$ the corresponding sequence $T(b) = \langle t_1, t_2, \ldots \rangle$ is growing rapidly: By the equations in (2), we have $r_{i-1} \leq 1/(t_i - 1)$ and $1/t_{i+1} < r_i = r_{i-1} - 1/t_i$. Consequently, $1/t_{i+1} < 1/(t_i - 1) - 1/t_i$. Rewriting this yields the inequality $t_{i+1} > t_i(t_i - 1)$, which in turn is equivalent to

$$t_{i+1} - 1 \;\geq\; t_i(t_i - 1) \qquad \text{for all } i \geq 1. \tag{5}$$

Next, consider some fixed index $j \geq 1$. A straightforward inductive argument based on (5) yields that $t_{j+k} - 1 \geq (t_j - 1)^{k+1}$ holds for all $k \geq 0$. From this we get that

$$\sum_{i=j}^{\infty} \frac{1}{t_i - 1} \;=\; \sum_{k=0}^{\infty} \frac{1}{t_{j+k} - 1} \;\leq\; \sum_{k=0}^{\infty} (t_j - 1)^{-k-1} \;=\; \frac{1}{t_j - 2}. \tag{6}$$

For $j = 1$ this inequality demonstrates that the infinite series in equation (4) indeed converges, and that the function $\rho(b)$ is well-defined.

The following result will be used in the proof of Lemma 6.

Lemma 3. *Let* $z \geq 1$ *be an integer. Then the sequence* $T(b)$ *fulfills the inequality*

$$\frac{t_z + 1}{t_z} \cdot \sum_{i=z}^{\infty} \frac{1}{t_i} \;\leq\; \sum_{i=z}^{\infty} \frac{1}{t_i - 1}. \tag{7}$$

Proof. Omitted in this version. □

4 Some Properties of the Function $\rho(b)$

This section is devoted to the proof of Lemma 1. Since by (5) the underlying series converges fast, the values $\rho(1)$ and $\rho(2)$ in statement (i) of Lemma 1 are easy to approximate by a computer program. For statement (ii), consider an integer $m \geq 2$. Since the sequence $T(m)$ starts with $t_1 = m + 1$, the definition of $\rho(b)$ in (4) immediately yields $\rho(m) \geq 1/m$. Moreover, by setting $j = 1$ in inequality (6) we get that

$$\rho(m) \;=\; \sum_{i=1}^{\infty} \frac{1}{t_i - 1} \;\leq\; \frac{1}{t_1 - 2} \;\leq\; \frac{1}{m-1} \qquad \text{for all integers } m \geq 2. \tag{8}$$

This completes the proof of statement (ii). We turn to statement (iii). Let $1 \leq a < b$, and let $T(a) = \langle t_i \rangle$ and $T(b) = \langle t'_i \rangle$ denote the two infinite sequences associated with a and b. Define $j \geq 1$ to be the smallest index with $t_j \neq t'_j$. Since $a < b$, this implies $t_j \leq t'_j - 1$. Then

$$\rho(a) - \rho(b) \;=\; \sum_{i=j}^{\infty} \frac{1}{t_i - 1} - \sum_{i=j}^{\infty} \frac{1}{t'_i - 1} \;>\; \frac{1}{t_j - 1} - \frac{1}{t'_j - 2} \;\geq\; 0 \tag{9}$$

where we used (6) to derive the first inequality and $t_j \leq t'_j - 1$ in the second inequality. Hence $a < b$ indeed implies $\rho(a) > \rho(b)$.

Next, we turn to statement (iv). Let $m \geq 2$ be an integer and consider the value $b_m = 2m/(m + 2)$. It can be verfied that the series $T(b_m)$ starts with the term $t_1 = 2$, which is followed by the all the terms of the sequence $T(m)$. Consequently, $\rho(b_m) = 1 + \rho(m)$ holds and from (8) we get that $1 + 1/m \leq \rho(b_m) \leq 1+1/(m-1)$. As m goes to ∞, b_m tends to 2 from below, and $\rho(b_m)$ tends to 1 from above. Since $\rho(b)$ is a decreasing function by statement (iii), we have thus proved the first part of statement (iv). The second part of statement (iv) follows by combining statements (ii) and (iii).

The (very technical) proofs of statements (v) and (vi) will be given in the full version of this paper.

5 Proof of the Upper Bound

In this section, we prove the upper bound stated in Theorem 2. As usual, let $b \geq 1$ denote the bin size, and let $T(b) = \langle t_1, t_2, \ldots \rangle$ be the integer sequence associated with b. Let $\ell \geq 3$ be an integer. We introduce t_ℓ intervals $\mathcal{I}_j$ with $j = 1, \ldots, t_\ell$ that form a partition of the interval $(0, b]$. For $1 \leq j \leq t_\ell - 1$, we define the interval $\mathcal{I}_j = (\frac{b}{j+1}, \frac{b}{j}]$. Moreover, we define the last interval $\mathcal{I}_{t_\ell} = (0, b/t_\ell]$.

Our online algorithm keeps one active bin $\mathcal{B}_j$ for every interval $\mathcal{I}_j$ ($j = 1, \ldots, t_\ell$). All items from the interval $\mathcal{I}_j \cap (0, 1]$ are packed into the corresponding active bin $\mathcal{B}_j$. If a newly arrived item does not fit into $\mathcal{B}_j$, this bin is closed, and a new corresponding bin for interval $\mathcal{I}_j$ is opened. In other words, the items from

interval $\mathcal{I}_j \cap (0,1]$ are packed into the active bins $\mathcal{B}_j$ according to the NEXT-FIT algorithm. This completes the description of the online algorithm.

To analyze this online algorithm, we define the following *weight function* $w : (0,1] \to \mathbb{R}$. For items x in $\mathcal{I}_j$ with $1 \le j \le t_\ell - 1$, we define $w(x) = 1/j$. For items x in the last interval $\mathcal{I}_{t_\ell}$, we define $w(x) = (xt_\ell)/(bt_\ell - b)$. The weight of a packed bin equals the sum of the weights of the items contained in this bin. The weight $w(L)$ of an item list L equals the sum of the weights of the items in L.

Lemma 4. *Every bin of size b that has been closed by the online algorithm contains items of total weight at least* 1.

Proof. First assume that the closed bin belongs to an interval $\mathcal{I}_j$ with $1 \le j \le t_\ell - 1$. Then it contains exactly j items, and each of these items has weight $1/j$. Next assume that the closed bin belongs to the interval $\mathcal{I}_{t_\ell}$. Then the bin has been closed, since a new item from $\mathcal{I}_{t_\ell}$ did not fit into it. Hence, the total size of its items is at least $b - b/t_\ell$. Since on the interval $\mathcal{I}_{t_\ell}$ the weight function is linear with slope $t_\ell/(bt_\ell - b)$, the weight of such a bin is at least 1. □

Lemma 5. *Let $1 \le z \le \ell - 1$ be an integer. Then for every positive real number $x \le b/t_z$, we have $w(x)/x \le (t_z + 1)/(bt_z)$.*

Proof. Omitted in this version. □

Lemma 6. *In any packing of the list L into unit-size bins, every unit-size bin receives items of total weight at most*

$$\sum_{i=1}^{\ell} \frac{1}{t_i - 1} + \frac{1}{(t_\ell - 1)^2}. \tag{10}$$

Proof. Consider some fixed unit-size bin $\mathcal{B}$ that contains the items $f_1 \ge f_2 \ge \cdots \ge f_n$ with total size at most 1. We distinguish three cases.

(Case 1) For $i = 1, \ldots, \ell$ we have $f_i \in (b/t_i, b/(t_i - 1)]$. We denote by F the sum of the sizes of the remaining items f_i with $i > \ell$. By the definition of the values t_i in (1) and (2), we conclude that

$$F \;=\; \sum_{i=\ell+1}^{n} f_i \;\le\; 1 - \sum_{i=1}^{\ell} \frac{b}{t_i} \;=\; b \cdot r_\ell \;\le\; \frac{b}{t_{\ell+1} - 1}. \tag{11}$$

Hence, all items $f_{\ell+1}, \ldots, f_n$ are in the last interval $\mathcal{I}_{t_\ell}$. By the definition of the weight function, the weight of the bin $\mathcal{B}$ then is upper bounded by

$$\begin{aligned}\sum_{i=1}^{\ell} \frac{1}{t_i - 1} + \frac{t_\ell}{bt_\ell - b} F &\le \sum_{i=1}^{\ell} \frac{1}{t_i - 1} + \frac{t_\ell}{(t_\ell - 1)(t_{\ell+1} - 1)} \\ &\le \sum_{i=1}^{\ell} \frac{1}{t_i - 1} + \frac{1}{(t_\ell - 1)^2}.\end{aligned}$$

Here we used (11) to derive the first inequality, and (5) to derive the second inequality. This completes the analysis of the first case.

(Case 2) There exists an integer z with $1 \le z \le \ell - 1$ such that the following holds: For $i = 1, \ldots, z-1$ we have $f_i \in (b/t_i, b/(t_i - 1)]$. Moreover, f_z either does not exist (since $n = z - 1$ holds) or if it does exist then $f_z \notin (b/t_z, b/(t_z - 1)]$ holds. We denote by F the sum of the sizes of the remaining items f_i with $i \ge z$. Similarly as above, we observe that

$$F \;=\; \sum_{i=z}^{n} f_i \;\le\; 1 - \sum_{i=1}^{z-1} \frac{b}{t_i} \;=\; \sum_{i=z}^{\infty} \frac{b}{t_i}. \tag{12}$$

By combining (12) with (6) we get that the total size F of all items $f_z, \ldots, f_n$ is at most $b/(t_z - 1)$. Since the largest one of all these items, f_z, is not contained in the interval $(b/t_z, b/(t_z - 1)]$, we conclude that the size of every item $f_z, \ldots, f_n$ is at most b/t_z. Then by Lemma 5, their overall weight is at most $F(t_z + 1)/(bt_z)$. The weight of the bin $\mathcal{B}$ is at most

$$\begin{aligned}\sum_{i=1}^{z-1} \frac{1}{t_i - 1} + \frac{F(t_z+1)}{bt_z} &\le \sum_{i=1}^{z-1} \frac{1}{t_i - 1} + \frac{t_z + 1}{t_z} \sum_{i=z}^{\infty} \frac{1}{t_i} \;\le\; \sum_{i=1}^{\infty} \frac{1}{t_i - 1} \\ &\le \sum_{i=1}^{\ell} \frac{1}{t_i - 1} + \frac{1}{t_{\ell+1} - 2} \;\le\; \sum_{i=1}^{\ell} \frac{1}{t_i - 1} + \frac{1}{(t_\ell - 1)^2}.\end{aligned}$$

Here we have first applied (12) to bound F from above, then the statement in Lemma 3, then the inequality in (6) to bound $\sum_{i=\ell+1}^{\infty} 1/(t_i - 1)$ from above, and in the end the inequality (5) together with $t_\ell \ge 2$. This completes the analysis of the second case.

(Case 3) This case is essentially the second case with $z = \ell$, which needs special treatment since the statement in Lemma 5 does not carry over to $z = \ell$. Assume that for $i = 1, \ldots, \ell - 1$ we have $f_i \in (b/t_i, b/(t_i - 1)]$, and that $f_\ell \notin (b/t_\ell, b/(t_\ell - 1)]$; the subcase where f_ℓ does not exist is trivial. We denote by F the sum of the sizes of the items f_i with $i \ge \ell$.

$$F \;=\; \sum_{i=\ell}^{n} f_i \;\le\; 1 - \sum_{i=1}^{\ell-1} \frac{b}{t_i} \;=\; b \cdot r_{\ell-1} \;\le\; \frac{b}{t_\ell - 1}. \tag{13}$$

Consequently, all items $f_\ell, \ldots, f_n$ are contained in the last interval $\mathcal{I}_{t_\ell}$. Then the weight of the bin $\mathcal{B}$ is at most

$$\sum_{i=1}^{\ell-1} \frac{1}{t_i - 1} + \frac{t_\ell}{bt_\ell - b} F \;\le\; \sum_{i=1}^{\ell-1} \frac{1}{t_i - 1} + \frac{t_\ell}{(t_\ell - 1)^2} \;=\; \sum_{i=1}^{\ell} \frac{1}{t_i - 1} + \frac{1}{(t_\ell - 1)^2}.$$

Here we used (13) to bound F. This completes the proof. □

Theorem 7. *For any bin size $b > 1$ and for any real $\varepsilon > 0$, there exist a sufficiently large k and an online k-bounded space bin packing algorithm A with $R_b(A) \le \rho(b) + \varepsilon$.*

Proof. Choose a sufficiently large integer $\ell \geq 3$ such that $1/(t_\ell - 1)^2 \leq \varepsilon$ is fulfilled. Then we derive from Lemma 4 that $A_b(L) \leq w(L)$, and we derive from Lemma 6 that $w(L) \leq (\rho(b) + \varepsilon) \cdot \textsc{Opt}_1(L)$. □

6 Proof of the Lower Bound

In this section, we prove the lower bound stated in Theorem 2. Consider an arbitrary online k-bounded space algorithm A for bin packing with bin size b. Let $T(b) = \langle t_1, t_2, \ldots \rangle$ be the integer sequence associated with b. Let ℓ be an integer, and let $\varepsilon > 0$ be a small real number such that $\varepsilon \cdot t_{\ell+1} \cdot \ell \leq 1$. Furthermore, let $N > k^3 t_{\ell+1}$ be a huge integer. We confront the online algorithm with several phases of 'bad' items, and we show that algorithm A eventually must perform poorly.

Alltogether there are ℓ phases. In the jth phase ($j = 1, \ldots, \ell$), exactly N items of size $b/t_{\ell-j+1} + \varepsilon$ arrive. The best that the bounded space algorithm A can do is to pack these items together in groups of cardinality $t_{\ell-j+1} - 1$ each. This consumes $N/(t_{\ell-j+1} - 1)$ bins. At the beginning of a phase up to k used bins of the previous phase are active, and this may save up to k bins. Summarizing, algorithm A uses at least $N/(t_{\ell-j+1} - 1) - k$ bins for packing the items of phase j. Adding this up over all $j = 1, \ldots, \ell$, we get that

$$A_b(L) \;\geq\; \sum_{j=1}^{\ell} \left(\frac{N}{t_{\ell-j+1} - 1} - k \right) \;=\; N \cdot \sum_{j=1}^{\ell} \frac{1}{t_j - 1} - k\ell. \tag{14}$$

By (3) and by the choice of ε, the ℓ items $b/t_{\ell-j+1} + \varepsilon$ with $1 \leq j \leq \ell$ together fit into a bin of size 1. Consequently, we have $\textsc{Opt}_1(L) \leq N$. By making N sufficiently large, (14) yields that the worst case performance $R_b(A)$ of algorithm A is at least $\sum_{j=1}^{\ell} \frac{1}{t_j - 1}$. Since this statement holds true for every value of ℓ, we may make ℓ arbitrarily large and thus make this bound arbitrarily close to $\rho(b)$.

Theorem 8. *For any $b \geq 1$ and for any online k-bounded space bin packing algorithm A, we have $R_b(A) \geq \rho(b)$.* □

Acknowledgements

We thank Clemens Heuberger for several discussions, and we thank Bettina Klinz for helping us in generating the picture.

References

1. S. Albers, S. Arora, and S. Khanna [1999]. Page replacement for generalized caching problems. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*, 31–40.
2. J. Csirik and G.J. Woeginger [1998]. Online packing and covering problems. In *Online Algorithms: The State of the Art*, LNCS 1442, Springer Verlag, 147–178.

3. J. Edmonds [1999]. Scheduling in the dark. In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing (STOC'99)*, 179–188.
4. B. Kalyanasundaram and K. Pruhs [1995]. Speed is more powerful than clairvoyance. In *Proceedings of the 36th IEEE Symposium on Foundations of Computer Science (FOCS'95)*, 214–221.
5. B. Kalyanasundaram and K. Pruhs [1995]. The online transportation problem. In *Proceedings of the 3rd European Symposium on Algorithms (ESA'95)*, Springer LNCS 979, 484–493.
6. C.C. Lee and D.T. Lee [1985]. A simple online bin-packing algorithm. *Journal of the ACM 32*, 562–572.
7. I.M. Niven and H.S. Zuckerman [1960]. An introduction to the theory of numbers. John Wiley.
8. C.A. Phillips, C. Stein, E. Torng, and J. Wein [1997]. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC'97)*, 140–149.
9. S. Wagon [1991]. Mathematica in action. W.H. Freeman.

Optimal Projective Algorithms for the List Update Problem

Christoph Ambühl[1], Bernd Gärtner[1], and Bernhard von Stengel[2]

[1] Institute for Theoretical Computer Science, ETH Zürich, 8092 Zürich, Switzerland.
{ambuehl, gaertner}@inf.ethz.ch
[2] Mathematics Department, London School of Economics, London WC2A 2AE, United Kingdom
stengel@maths.lse.ac.uk

Abstract. The list update problem is a classical online problem, with an optimal competitive ratio that is still open, somewhere between 1.5 and 1.6. An algorithm with competitive ratio 1.6, the smallest known to date, is COMB, a randomized combination of BIT and TIMESTAMP. This and many other known algorithms, like MTF, are *projective* in the sense that they can be defined by only looking at any pair of list items at a time. Projectivity simplifies both the description of the algorithm and its analysis, and so far seems to be the only way to define a good online algorithm for lists of arbitrary length. In this paper we characterize all projective list update algorithms and show their competitive ratio is never smaller than 1.6. Therefore, COMB is a best possible projective algorithm, and any better algorithm, if it exists, would need a non-projective approach.

1 Introduction

The *list update problem* is a classical online problem in the area of self-organizing data structures [4]. Requests to items in an unsorted linear list must be served by accessing the requested item. We assume the *partial cost model* where accessing the ith item in the list incurs a cost of $i -$ units. This is simpler to analyze than the original *full cost model* [12] where that cost is i. The goal is to keep access costs small by rearranging the items in the list. After an item has been requested, it may be moved free of charge closer to the front of the list. This is called a *free exchange*. Any other exchange of two consecutive items in the list incurs cost one and is called a *paid exchange*.

An *online* algorithm must serve the sequence σ of requests one item at a time, without knowledge of future requests. An optimum *offline* algorithm knows the entire sequence σ in advance and can serve it with minimum cost $OFF\ \sigma$. If the online algorithm serves σ with cost $ON\ \sigma$, then it is called *c-competitive* if for a suitable constant b

$$ON\ \sigma\ \leq c \cdot OFF\ \sigma\ + b$$

for all request sequences σ. The *competitive ratio* c in this inequality is the standard yardstick for measuring the performance of the online algorithm. The well-known *move-to-front* rule MTF, for example, which moves each item to the front of the list after it has been requested, is 2-competitive [12,13]. This is also the best possible competitiveness

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 305–316, 2000.
© Springer-Verlag Berlin Heidelberg 2000

for any deterministic online algorithm for the list update problem [12]. Another deterministic algorithm that is also 2-competitive is TIMESTAMP due to Albers [1], which moves the requested item x in front of all items which have been requested at most once since the last request to x.

Randomized algorithms can perform better on average, as first shown by Irani [10,11]. Such an algorithm is called c-competitive if

$$E\left[ON\ \sigma\ \right] \leq c \cdot OFF\ \sigma\ + b,$$

where the expectation is taken over the randomized choices of the online algorithm. Randomization is useful only against the *oblivious adversary* [6] that generates request sequences without observing the randomized choices of the online algorithm. If the adversary can observe those choices, it can generate requests as if the algorithm was deterministic, which is then at best 2-competitive. We therefore consider only the interesting situation of the oblivious adversary.

In this case, lower bounds for the competitive ratio are harder to find; the first nontrivial bounds are due to Karp and Raghavan, see the remark in [12]. A general technique is Yao's theorem [15]: If there is a probability distribution on request sequences so that the resulting *expected competitive ratio* for any deterministic online algorithm is d or higher, then every deterministic or randomized online algorithm has competitive ratio d or higher [9]. In the partial cost model, a lower bound of 1.5 is easy to find as only two items are needed. Teia [14] generalized this idea to prove the same bound in the full cost model, where long lists are needed. Ambühl, Gärtner and von Stengel [5] showed a lower bound of 1.50084 for lists with five items in the partial cost model, using game trees and a modification of Teia's approach. The optimal competitive ratio for the list update problem (in the partial cost model) is therefore between 1.50084 and 1.6, but the true value is as yet unknown.

The upper bound of 1.6 is the last link so far in a chain of results, starting with the observation that MTF acts too eagerly in moving items to the front. The better algorithms BIT, COUNTER, and RANDOM RESET move the requested item to the front or leave it at its position, depending on the number of previous requests to the currently requested item. The elegant BIT algorithm stores a data bit—initially set to a random value—with each item. The bit is flipped at each request, and the item is moved to the front of the list when the bit has been set to one. BIT is 1.75-competitive. The related RANDOM RESET algorithm has competitive ratio $\sqrt{3}$, about 1.73. This ratio is improved to the Golden Ratio $\ + \sqrt{5}\ /\ $, about 1.62, by treating each item with a randomized combination of TIMESTAMP and MTF [1].

The best randomized list update algorithm known to date is the 1.6-competitive algorithm COMB [2]. It serves the request sequence with probability 4/5 using BIT [12]. With probability 1/5, COMB treats the request sequence using TIMESTAMP.

With the exception of Irani's algorithm SPLIT [10,11], all the specific list update algorithms mentioned above are *projective*, meaning that the relative order of any two items in the list only depends on previous requests to those items. (A simple example for a non-projective algorithm is TRANSPOSE, which moves the requested item just one position further to the front.) The main result of this paper is a proof that, surprisingly, 1.6 is the best possible competitive ratio attainable by a projective algorithm. As a tool,

we develop an explicit characterization of deterministic projective algorithms in terms of two functions for every item, responsible for the "macro"- and the "micro"-behavior of the item.

This result is significant in several respects. First, it puts an end to the search for improved algorithms via combinations of existing projective ones. This approach has been used successfully in the past, as the results mentioned above indicate, but it has reached its limits with the development of the COMB algorithm. New and better algorithms (if they exist) have to be non-projective, and must derive from new, yet to be discovered, design principles. Second, the characterization of projective algorithms is a step forward in understanding the structural properties of list update algorithms. Under this characterization, the largest and so far most significant class of algorithms appears in a new, unified way. Third, our lower bound construction gives rise to an explicit test scenario for new algorithms: we construct a set of request sequences with the property that a randomly chosen instance from the set is "hard" for any projective algorithm. A new, supposedly better, algorithm should therefore be able to defeat those hard instances, and this might be more difficult than to defeat some ad-hoc set of instances.

2 Projective Algorithms

Consider a list with n items. For a request sequence σ and two list items x and y, the *projection* of σ on the unordered pair $\{x, y\}$ is denoted by σ_{xy} and defined as the sequence obtained from σ by deleting all requests to items other than x or y. We write σ_x instead of σ_{xx}. For a given deterministic online algorithm, let $S\ \sigma$ denote the list state after the request sequence σ is served. List states are written as $[x_1 x_2 \ldots x_n]$ where x_1 is the item at the front of the list. The list state projected to the pair $\{x, y\}$ is denoted by $S_{xy}\ \sigma$, which is either $[xy]$ or $[yx]$, indicating the relative position of x and y after σ is served.

A deterministic algorithm is projective if the relative order of any two items after any sequence σ does not depend on requests to the other items:

Definition 1 (Projective Algorithms). *A deterministic list update algorithm is projective if for all request sequences σ and any two list items x and y*

$$S_{xy}\ \sigma = S_{xy}\ \sigma_{xy}\ . \tag{1}$$

A projective algorithm A can be analyzed in a much simpler way than a general one, since only two-item lists have to be studied [7]. If the projected cost on two items x, y of this algorithm is A_{xy} (with each request to x or y contributing either 0 or 1, depending on whether the requested item is before or behind the other item in the list), then its total cost is given by

$$A\ \sigma = \sum_{\{x,y\} \subseteq L} A_{xy}\ \sigma_{xy}\ , \tag{2}$$

where L is the set of list items [2,8]. Furthermore, the optimal offline cost OFF_{xy} of serving σ_{xy} is easy to describe, and gives a lower bound $\overline{OFF}\ \sigma$ defined similar to (2)

for the optimal offline cost. This quantity is used to prove the competitive ratio of COMB and other algorithms. The simple equation (2) holds only in the partial cost model, which is therefore the natural choice when studying projective algorithms.

Projective algorithms have a natural generalization, where we demand the relative order of any k-tuple of list items to depend only on the requests to these k items. It turns out that for lists with more than k items, only projective algorithms satisfy this condition. This follows from the fact that e.g. for $k = 3$, $S_{xyz}\ \sigma\ = S_{xyz}\ \sigma_{xyz}$ implies that the relative order of any pair from $\{x, y, z\}$ is independent of the requests to any item in $L \setminus \{x, y, z\}$. As soon as we have $k +$ list items, it is easy to see that the constraints for all k-tuples enforce the relative order of any pair of list items to be independent of the requests to other items.

We define a *randomized* online algorithm as projective if it is a (not necessarily finite) probability distribution over deterministic projective algorithms. A less restrictive definition is conceivable, but would not allow us to prove the lower bound for projective algorithms that we intend and that we think is useful. Namely, one could call a randomized online list update algorithm projective if serving any request sequence σ induces a distribution on list states $S_{xy}\ \sigma$ that only depends on σ_{xy}. To illustrate the problem with this definition, consider the following randomized algorithm on a list of two items only: If the current list state is $[xy]$ and y is requested following a request to x, move y to the front with probability 1/2, and if y is requested following a request to y (that is, y was not moved at the preceding request), then move y to the front with certainty. It is not hard to see that such a randomized online algorithm has competitive ratio 1.5, which is the best possible. Furthermore, any randomized algorithm showing this as projective behavior on a longer list would also be 1.5-competitive. It is indeed possible to construct such an algorithm for lists with up to four items using partial orders [3], but impossible for lists with five or more items, as recently proved by the authors [5].

In the following, we will characterize the deterministic projective algorithms in a way that makes their projective behavior transparent, and unifies many known algorithms. By our above assumption that considers a randomized projective algorithm as a probability distribution over deterministic ones, we will be able to use this characterization in the lower bound proof later.

3 Critical Requests

Consider a given deterministic online list update algorithm that is projective according to Definition 1. In order to obtain a meaningful characterization of such an algorithm, we assume $n >$ since on lists with only two items, any algorithm is projective. Let $i, j > 0$ and consider request sequences σ with exactly i requests to x and j requests to y ($x \neq y$), that is, $\sigma_x = x^i$ (the i-fold repetition of x) and $\sigma_y = y^j$. Then we say x^i and y^j are *equivalent*, written $x^i \sim y^j$, if there are request sequences σ and σ' so that

$$\begin{aligned} \sigma_x = \sigma'_x = x^i, \quad \sigma_y = \sigma'_y = y^j, \\ S_{xy}\ \sigma\ = [xy], \quad S_{xy}\ \sigma'\ = [yx]. \end{aligned} \tag{3}$$

In other words, one should be able to shuffle the requests to x and y such that either x or y is in front after serving the request sequence. Assume, for the moment, that (3)

holds for any two items x and y and any $i, j > 0$ (we deal with the general case in the next section). Under this assumption, the algorithm can be characterized in terms of the central concept of *critical requests*.

For any list item x, let $F_x \quad \mathrm{N}^+ \to \mathrm{N}^+$ be a function so that $F_x \ i \ \leq i$ for all i. Then F_x defines the *critical request* to x in a request sequence σ as the $F_x \ i$ th request to x in σ if $\sigma_x = x^i$. We say that the given algorithm *operates* according to these critical requests if, after serving any request sequence σ, the relative order of the items requested at least once is the reverse order of their critical requests in σ. In other words, x precedes y after σ with $\sigma_x = x^i$, $\sigma_y = y^j$, and $i, j > 0$ if and only if the $F_x \ i$ th request of x was later than the $F_y \ j$ th request to y. As a simple illustration, observe that MTF uses $F_x \ i \ = i$ for all $i > 0$ and $x \in L$. In the next section we will also deal with non-requested items. Paid exchanges can be represented by critical request functions that are not monotone.

There is also a natural "dual" way to deal with critical requests: At any time, an item precedes another if its critical request was *earlier*. In this case, we say that the algorithm operates *dually* according to critical requests. For example, operating dually on $F_x \ i \ = i$, for all i, results in the *move-to-back* algorithm. Although such behavior cannot be competitive, it defines a projective algorithm.

Any critical request functions F_x for the list items x therefore define two list update algorithms. The algorithms are projective since $F_x \ i \ $ does not depend on requests to items other than x. Furthermore, condition (3) holds for $i, j > 0$: sequences σ and σ' with $\sigma_{xy} = x^i y^j$ and $\sigma'_{xy} = y^j x^i$ will always result in $S_{xy} \ \sigma \ \neq S_{xy} \ \sigma \ $. The following theorem shows that, conversely, *any* projective list update algorithm fulfilling (3) arises from critical requests.

Theorem 1. *Let A be a projective algorithm on a list with n items, $n >$, so that for all items x, y and all $i, j > 0$, property (3) holds. Then A operates (or operates dually) according to suitable critical request functions.*

Proof. Assume that $i, j, k > 0$ and σ and σ' are request sequences (over only three items x, y and z) with $\sigma_x = \sigma'_x = x^i$, $\sigma_y = \sigma'_y = y^j$, and $\sigma_z = \sigma'_z = z^k$, so that

$$S_{xy} \ \sigma_{xy} \ = [xy] \quad \text{and} \quad S_{xz} \ \sigma_{xz} \ = [zx] \tag{4}$$

and

$$S_{xy} \ \sigma'_{xy} \ = [yx] \quad \text{and} \quad S_{xz} \ \sigma'_{xz} \ = [xz]. \tag{5}$$

Such sequences exist by assumption (3) and projectivity of A, and since the projections considered in these equations can be combined into sequences σ and σ' of requests to the three items.

Note that (4) and (5) imply $S_{yz} \ \sigma \ = [zy]$ and $S_{yz} \ \sigma' \ = [yz]$, hence by projectivity

$$\sigma_{yz} \neq \sigma'_{yz}. \tag{6}$$

By labelling each request to an item with its position in the unary projection to that item (e.g. the fifth request to x will be labelled $x_{(5)}$), σ_{xy} and σ'_{xy} (and similarly σ_{xz} and σ'_{xz}) can be considered as permutations that may be transformed into each other by successively transposing pairs of consecutive requests.

We can even assume that σ_{xy} and σ'_{xy} differ only in a single such transposition, namely the (not necessarily unique) one responsible for the reversal of the list state S_{xy} during the transformation. Similarly, we assume that σ_{xz} and σ'_{xz} differ only in a single transposition of consecutive requests to x and z.

Suppose the transposition $\sigma_{xy} \to \sigma'_{xy}$ involves $x_{(q)}$ and $y_{(\ell)}$, while $x_{(r)}$ and $z_{(m)}$ participate in the transposition $\sigma_{xz} \to \sigma'_{xz}$. We now prove that $q = r$, which is also easily seen to imply that this value is well-defined: It neither depends on σ and σ' nor on the specific transposition we consider, but only on σ_x.

For this, assume $q \neq r$ and consider the sequence σ. W.l.o.g., $x_{(q)}$ and $y_{(\ell)}$ are consecutive requests in σ—otherwise we can transpose $y_{(\ell)}$ with all in-between requests to z without changing the projections to $\{x, y\}$ and $\{x, z\}$. Similarly, suppose that $x_{(r)}$ and $z_{(m)}$ are consecutive. A sequence σ' satisfying (5) is now obtained from σ by transposing both $x_{(q)}$ with $y_{(\ell)}$ and $x_{(r)}$ with $z_{(m)}$. Under this operation, however, the projection to $\{y, z\}$ remains invariant, a contradiction to (6). Hence, we must have $q = r$.

We have seen that for all items x and all i, there is a unique critical value $F_x\ i\ = q$, and it remains to show that A operates (or operates dually) according to the $F_x\ i\ $.

First of all, we need to show that the relative order of any two items only depends on the order of their critical requests. By the above arguments, whenever σ_{xy} and σ'_{xy} satisfy

$$S_{xy}\ \sigma_{xy}\ \neq S_{xy}\ \sigma'_{xy} \tag{7}$$

and differ only by a single transposition of consecutive requests, this transposition involves the critical request of x. By symmetry, it also involves the critical request of y. In general, when we transform σ_{xy} into σ'_{xy} by transposing consecutive requests, property (7) holds if and only if the two critical requests have been transposed an odd number of times, which fixes their relative order.

Now consider a request sequence σ over an n-item list such that $S\ \sigma\ = [x_1 x_2 \ldots x_n]$. Let p_i be the position of x_i's critical request in σ. If we do not have $p_1 > p_2 > \cdots > p_n$ (A operates on F) or $p_1 < p_2 < \cdots < p_n$ (A operates dually on F), we must have an index i such that either $p_i < p_{i+1} > p_{i+2}$ or $p_i > p_{i+1} < p_{i+2}$. In both cases, we can manipulate σ such that the critical requests of x_i and x_{i+2} change their order, but both keep their relative order w.r.t. the critical request of x_{i+1}. In the list obtained after serving σ, items x_i and x_{i+2} change their relative order under this manipulation, while they keep their relative order w.r.t. x_{i+1}. This is impossible. □

The assumption of at least three list items in the preceding theorem is crucial. On lists with only two items, any algorithm is projective, but cannot always be defined in terms of critical requests. This follows from cardinality considerations: There are $\binom{i+j}{i}$ request sequences with i requests to x and j requests to y, each of which can have its own list state after being served, but only $i \cdot j$ many ways of defining critical requests. As an example, the algorithm that puts the second-to-last requested item at the front of the two-item list cannot be defined in terms of critical requests.

4 Containers

Not all projective algorithms fulfill condition (3) for all x^i and y^j. As an example, consider the algorithm where all items requested an odd number of times precede all items requested an even number of times, and where the items within each of these two sets are arranged according to the MTF rule. Then (3) fails if i is odd and j is even or vice versa. According to the general characterization of projective algorithms that we will give in this section, the odd- and even-requested items in this example form separate "containers" that represent sublists of the list. Within one container, items are moved according to critical requests, but items in different containers are always in one and the same relative position.

To make this precise, consider a given projective list update algorithm and the set

$$U = \{x^i \mid i \in \mathrm{N},\ x \in L\}$$

of unary projections of request sequences, where L denotes the set of items in the list.

Recall that we write $x^i \sim y^j$ whenever (3) holds. It makes sense to allow $i = 0$ and $j = 0$ as well. By generalizing (3) we get $x^0 \not\sim y^j$ for all $x \neq y$ and $j \in \mathrm{N}$.

For x, y, z distinct, if $x^i \sim y^j$ and $y^j \sim z^k$, then, by projectivity, there are always two sequences σ and σ' containing the three unary projections such that $S\ \sigma\ = [xyz]$ and $S\ \sigma'\ = [zyx]$, which implies $x^i \sim z^k$. It is easy to see that $\sim$ is an *equivalence relation* on U if we stipulate $x^i \sim x^i$ for any $x^i \in U$ and also $x^i \sim x^j$ for $i \neq j$ if and only if there is a z^k with $z \neq x$ such that $x^i \sim z^k$ and $z^k \sim x^j$.

An equivalence class under $\sim$ shall be called a *container*. Let $\mathcal{C}$ denote the set of containers and $c_x\ i$ denote the container containing x^i. If x^i and y^j are in different containers, we write $c_x\ i\ < c_y\ j$ whenever for some σ with $\sigma_x = x^i$ and $\sigma_y = y^j$ we have $S_{xy}\ \sigma\ = [xy]$ (and hence for all such σ, since (3) does not hold). It is easy to see that this does not depend on the choice of the representatives x^i and y^j from each container. A special case occurs if both x^i and x^j are the only members in their respective containers. In this case, no canonical order exists, and we set $c_x\ i\ < c_x\ j$ if $i < j$.

Then $<$ defines a *total order* on the containers, which has a natural interpretation: After a request sequence σ, consider the unary projections σ_x for each item x, and their corresponding containers. If two projections x^i and y^j are in different containers, x is in front of y if and only if $c_x\ i\ < c_y\ j$. Hence the containers represent sublists of the list state $S\ \sigma$, and we will say that $c_x\ i$ contains the item x if $\sigma_x = x^i$.

This characterizes any projective algorithm, apart from its behavior within each container, which is easy to describe: If there is only one item in the container, the position of the item is that of the container. A container containing only unary projections for two distinct items can have an arbitrary behavior of its items, since any algorithm on only two items is projective. If the container contains projections for at least three items, Theorem 2 applies, that is, the algorithm operates or operates dually according to suitable critical requests defined for the unary projections in that container. For this, observe that by definition of $\sim$, all empty projections x^0 are in a container of their own, so whenever a container has at least two items, each item has been requested at least once, in which case the critical requests exist.

To summarize, we can express any deterministic projective algorithm by a pair of functions $c_x \quad \mathrm{N} \to \mathcal{C}$ and $F_x \quad \mathrm{N}^+ \to \mathrm{N}^+$ for all $x \in L = \{x_1, x_2, \ldots, x_n\}$. The ordering of the values $c_x \ 0$ corresponds to the initial list state. In the following examples, we denote the containers by integers, where the ordering of the integers corresponds to the ordering of the containers.

MTF moves all items into a common container 0 at their first request. All items use $c_{x_i} \equiv \ i, 0, 0, 0, \ldots$ and $F_{x_i} \ k \ = k$.

TIMESTAMP moves all items into a common container 0 at their second request. By definition of $\sim$, an item cannot stay in its initial container after the first request, so all items will use $c_{x_i} \equiv \quad i, \ i - \ , 0, 0, 0, \ldots$ and $F_{x_i} \ k \ = \qquad , k - \quad .$

FREQUENCY COUNT makes heavy use of containers. Items are ordered according to the number of requests to them, so all items requested k times are in container $-k$. The functions used are $c_{x_i} \equiv \ i, - \ , - \ , \ldots$ and $F_{x_i} \ k \ = k$.

BIT is a randomized algorithm. In the beginning, every item tosses a fair coin to decide whether it uses the pair $F^0_{x_i}, c^0_{x_i}$ or $F^1_{x_i}, c^1_{x_i}$ with $c^0_{x_i} \equiv \quad i, \ i - \ , 0, 0, 0, \ldots$, $F^0_{x_i} \equiv \quad , \ , 3, 3, 5, 5, \ldots$, $c^1_{x_i} \equiv \quad i, 0, 0, 0, \ldots$, $F^1_{x_i} \equiv \quad , \ , \ , 4, 4, \ , \ , \ldots$.

5 Lower Bound

In this section, we use the characterization of projective algorithms from the previous section to prove that no such algorithm is better than 1.6-competitive. Algorithms with a good competitive ratio never operate dually according to critical requests, and have the critical request close to the last request, for every item. That is, they fulfill $i - F_x \ i \ \leq$ most of the time. Therefore we work with $f_x \ i \ = i - F_x \ i$ in the following. Recall that MTF is defined by $f_x \ i \ = 0$, and TIMESTAMP by $f_x \ i \ = \ .$

Motivated by this discussion, we consider projective algorithms A for lists of more than two items that fulfill the following additional assumptions:

(i) A constant p exists such that after at most p requests to every item, all items will reside in a single common container, and A operates (i.e. does not operate dually) according to critical requests within that container, and

(ii) the values $f_x \ i$ that determine the critical requests can be uniformly bounded by some constant M, where w.l.o.g. $M \geq 3$.

Let us call an algorithm satisfying (i) and (ii) *regular*.

Given any $\varepsilon > 0$ and b, we will show that there is a probability distribution π on a finite set Λ of request sequences so that

$$\sum_{\lambda \in \Lambda} \pi \ \lambda \ \frac{A \ \lambda}{OFF \ \lambda \ + b} \geq \ . \ - \varepsilon, \tag{8}$$

for any deterministic regular algorithm A. Then *Yao's theorem* [15] asserts that also any randomized regular algorithm has competitive ratio $\ . \ - \varepsilon$ or larger. This holds for any fixed p and M. Hence the competitive ratio is at least 1.6. This is achieved by COMB and therefore a tight bound for projective algorithms.

The same holds for general projective algorithms, but we defer the technicalities of that to the full paper (see also section 6).

All $\lambda \in \Lambda$ will consist of only two items x and y. With the constant M from (ii), let

$$\phi = x^M\ yxyx^M\ yx^M\ y^M\ xyxy^M\ xy^M\ x^M\ y^M. \tag{9}$$

ϕ consists of eight *blocks*, each of which ends in x^M or y^M. Let $H = |\phi|/\ \ $, and let k and T be arbitrary positive integers. Then the set of sequences in (8) is given by

$$\Lambda = \{x^t x^{3+h} y^{3+h} \phi^k \mid 0 \le h < H,\ 0 \le t < T\}, \tag{10}$$

where any λ in Λ is chosen with equal probability $\ /HT$ by π.

First, observe that *OFF* pays ten units for each repetition of ϕ (which always starts in offline list state $[yx]$), and therefore all sequences in Λ have equal offline cost $\ +\ 0k$. This and the fact that $\pi\ \lambda\ $ for $\lambda \in \Lambda$ is constant allows us to rewrite (8) as

$$\frac{\sum_{\lambda\in\Lambda} A\ \lambda}{\sum_{\lambda\in\Lambda} \mathit{OFF}\ \lambda\ + b} > \ .\ - \varepsilon. \tag{11}$$

The offline cost *OFF* $\lambda\ $ in (11), as well as the online cost $A\ \lambda\ $, can grow arbitrarily large with k, so that we can assume w.l.o.g. that $b = 0$ in (11), adapting ε suitably.

In the rest of this section we show that (10) yields the desired property (11). We say that A is in *state* $\ i, j\ $ if it has served σ with $\sigma_x = x^i$ and $\sigma_y = y^j$, where σ is some prefix of a sequence λ in Λ. That sequence σ is a random variable since the particular order of requests to x and y is usually not known.

We say that a sequence λ in Λ *switches from* x in state $\ i, j\ $ if λ has the prefix σ with $\sigma_x = x^i$ and $\sigma_y = y^j$ and σ ends in x^M. Similarly, we say that λ switches from y in state $\ i, j\ $ if that prefix σ ends in y^M. A state $\ i, j\ $ is called *good* if it fulfills the following conditions:

(a) there are four sequences in Λ that switch from x in state $\ i, j\ $. They continue with the requests y^M, y^M, yx^M, and $yxyx^M$, respectively;
(b) the same holds with x and y interchanged;
(c) properties (a) and (b) also hold for the states $\ i -\ \ , j\ $ and $\ i, j -\ \ \ .$

This means, for every good state $\ i, j\ $ and each of the eight blocks in ϕ, there is exactly one sequence σ in Λ that starts with this block in state $\ i, j\ $. Let $A\ i, j\ $ be the sum of costs incurred by A on those eight blocks, and *OFF* $\ i, j\ $ the corresponding sum of offline costs. In general, $A\ i, j\ $ denotes the sum of costs incurred by A on all next blocks of the (at most eight) sequences switching from x or y in state $\ i, j\ $, and *OFF* $\ i, j\ $ is the corresponding offline cost.

We will show that for every good state $\ i, j\ $, $A\ i, j\ $ is at least 16, while *OFF* $\ i, j\ $ is always 10. Together with the facts that most states are good states (which we will prove below), and that the cost for serving the initial prefix $x^t x^{3+h} y^{3+h}$ is independent of k

and can therefore be neglected for large k, this gives

$$\frac{\sum_{\lambda\in\Lambda} A\ \lambda}{\sum_{\lambda\in\Lambda} OFF\ \lambda} = \frac{\sum_{(i,j)} A\ i,j\ + \sum_{h,t} A\ x^t x^{3+h} y^{3+h}}{\sum_{(i,j)} OFF\ i,j\ + \sum_{h,t} OFF\ x^t x^{3+h} y^{3+h}}$$

$$\approx \frac{\sum_{(i,j)\ \text{good}} A\ i,j}{\sum_{(i,j)\ \text{good}} OFF\ i,j} \geq \min_{(i,j)\ \text{good}} \frac{A\ i,j}{OFF\ i,j},$$

thus proving the lower bound, because the minimum is at least $\ /\ 0 = \ .\ .$

By considering each of the four continuing sequences y^M, y^M, yx^M, and $yxyx^M$ in (a), we see that the sum of their offline costs is five. Hence, we have to show that the sum of online costs is at least eight. This is not always the case: It is possible that a certain choice of the critical request functions will result in an online cost of only seven units. However, we will show that those particular critical requests will incur nine units of online cost in state $\ i-\ ,j\ $, one of which we can "borrow" for $A\ i,j\ $. This results in eight units of online cost, for all good states.

Consider a sequence in a good state $\ i,j\ $ that, as in (a), switches from x, so that the next requests are y^M, y^M, yx^M, and $yxyx^M$ (our reasoning will then apply to (b) by symmetry). Since the first two of these requests are yy, yy, yx, yx, their total online cost is six units: By assumption (ii), the critical request to x is among the preceding requests x^M. Hence four units are to be paid for the first request to y, and then two extra units, namely on the request to x in yx and yx if $f_y\ j+\ \ = 0$, and on the second request to y in yy and yy if $f_y\ j+\ \ \geq\ $. A seventh unit is spent in serving the second request to y or to x in $yxyx^M$. The only case where no more than these seven online cost units occur is if

$$f_x\ i+\ \ = 0 \qquad \text{and} \qquad f_y\ j+\ \ =\ . \tag{12}$$

Namely, $f_y\ j+\ \ \leq\ $ is necessary since otherwise y would not be in front of x at the third request to y in the sequences y^M, adding at least two more cost units. If $f_y\ j+\ \ = 0$, then we need $f_x\ i+\ \ = 0$ to avoid another cost unit when serving the second request to x in yx^M, and hence $f_y\ j+\ \ \geq\ $ to avoid that y is moved to the front at the second request to y in $yxyx^M$ since that would create an extra cost unit for serving the second x. If $f_y\ j+\ \ \geq\ $, then we need again $f_x\ i+\ \ = 0$ and $f_y\ j+\ \ \geq\ $ to avoid that y is moved to the front at the second request to y in $yxyx^M$. Together, this implies (12). Hence, whenever (12) fails, the online algorithm incurs eight or more unit costs.

The seven online cost units in case (12) create *nine* online cost units in state $\ i-\ ,j\ $ for the sequences switching from y. Namely, by (c), the subsequent requests are x^M, x^M, xy^M, and $xyxy^M$. As before, six online units will be spent on the first two of these requests which are xx, xx, xy, xy, and a seventh unit either on the second y in xy^M or on the second x in $xyxy^M$. That last sequence, however, will incur two additional

cost units: Because of (12), x is in front of y after the third and forth request in $xyxy^M$, causing two more units for serving the second and third requests to y.

It remains to show that most states are good. The sequences λ in Λ are, by (10), essentially k-fold repetitions of ϕ.

The random initial subsequence $x^{3+h}y^{3+h}$ of λ means that all pairs of states i, j and $i+\ \ , j+\ \ \ \ $, apart from those with low or high values of i and j, are equally likely reached by any request in ϕ. Note that $|\phi_x| = |\phi_y|$. The additional prefix x^t of λ does the same for the pairs of states i, j and $i+\ \ , j$ except for those with small or large values of i and j. Figure 1 displays the good and the bad states in a diagram. There are $\Theta\ kHT$ good states, but only $\Theta\ kH^2$ bad ones. By choosing T large enough, the contribution of bad states can be neglected, so that (8) holds.

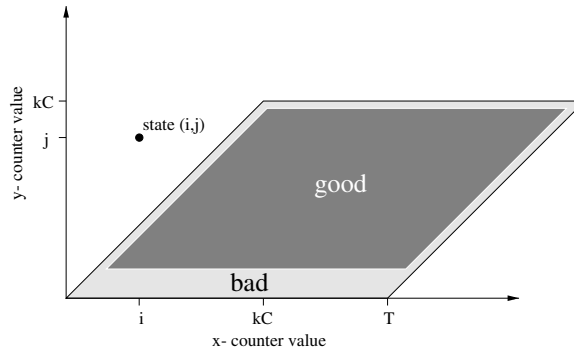


Fig. 1. xy-diagram

We have proved a lower bound of 1.6 for the competitive ratio of any *regular* projective algorithm A, defined by a probability distribution over deterministic regular algorithms. By definition, this means that A is using only one active container in the long run, and that the distance between current and critical request is bounded for all items. Because of lack of space, we will deal with the other cases only in the full version. We thus obtain

Theorem 2. *Any projective list update algorithm has a competitive ratio of at least . in the partial cost model, and this bound is best possible, as the algorithm COMB demonstrates.*

6 Conclusion

An open problem is to extend this result to the full cost model, even though this model is not very natural in connection with projective algorithms. This would require request sequences over arbitrarily many items, and it is not clear whether an approach similar to the one given here can work.

Another ambitious goal is to further improve the lower bound in case of non-projective algorithms. Here, the techniques of the paper do not apply at all, and to

get improvements that are substantially larger than the ones obtainable with the methods of [5] requires substantial new insights.

Finally, the search for good non-projective algorithms has become an issue with our result. Irani's SPLIT algorithm [10,11] is the only one known of this kind with a competitive ratio below 2. A major obstacle for finding such algorithms is the difficulty of their analysis, because pairwise methods are not applicable, and other methods (e.g. the potential function method) have not been studied in depth. We hope that our result can stimulate further research in this direction.

References

1. S. Albers (1998), Improved randomized on-line algorithms for the list update problem. *SIAM J. Comput.* 27, no. 3, 682–693 (electronic). Preliminary version in *Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms* (1995), 412–419.
2. S. Albers, B. von Stengel, and R. Werchner (1995), A combined BIT and TIMESTAMP algorithm for the list update problem. *Inform. Process. Lett.* 56, 135–139.
3. S. Albers, B. von Stengel, and R. Werchner (1996), List update posets. Manuscript.
4. S. Albers and J. Westbrook (1998), Self Organizing Data Structures. In A. Fiat, G. J. Woeginger, "Online Algorithms: The State of the Art", *Lecture Notes in Comput. Sci.*, 1442, Springer, Berlin, 13–51.
5. C. Ambühl, B. Gärtner, and B. von Stengel (2000), A new lower bound for the list update problem in the partial cost model. To appear in *Theoret. Comput. Sci.*
6. S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson (1994), On the power of randomization in on-line algorithms. *Algorithmica* 11, 2–14. Preliminary version in *Proc. 22nd STOC* (1990), 379–386.
7. J. L. Bentley, and C. C. McGeoch (1985), Amortized analyses of self-organizing sequential search heuristics. *Comm. ACM* 28, 404–411.
8. A. Borodin and R. El-Yaniv (1998), Online Computation and Competitive Analysis. Cambridge Univ. Press, Cambridge.
9. A. Borodin, N. Linial, and M. E. Saks (1992), An optimal online algorithm for metrical task systems. *J. ACM* 39, 745–763. Preliminary version in *Proc. 19th STOC* (1987), 373–382.
10. S. Irani (1991), Two results on the list update problem. *Inform. Process. Lett.* 38, 301–306.
11. S. Irani (1996), Corrected version of the SPLIT algorithm. Manuscript.
12. N. Reingold, J. Westbrook, and D. D. Sleator (1994), Randomized competitive algorithms for the list update problem. *Algorithmica* 11, 15–32.
13. D. D. Sleator, and R. E. Tarjan (1985), Amortized efficiency of list update and paging rules. *Comm. ACM* 28, 202–208.
14. B. Teia (1993), A lower bound for randomized list update algorithms, *Inform. Process. Lett.* 47, 5–9.
15. A. C. Yao (1977), Probabilistic computations: Towards a unified measure of complexity. *Proc. 18th FOCS*, 222–227.

Efficient Verification Algorithms for One-Counter Processes

Antonín Kučera*

Faculty of Informatics MU, Botanická 68a, 60200 Brno, Czech Republic, tony@fi.muni.cz

Abstract. We study the problem of strong/weak bisimilarity between processes of one-counter automata and finite-state processes. We show that the problem of weak bisimilarity between processes of one-counter nets (which are 'weak' one-counter automata) and finite-state processes is **DP**-hard (in particular, it means that the problem is both **NP** and **co-NP** hard). The same technique is used to demonstrate **co-NP**-hardness of strong bisimilarity between processes of one-counter nets. Then we design an algorithm which decides weak bisimilarity between processes of one-counter automata and finite-state processes in time which is polynomial for most 'practical' instances, giving a characterization of all hard instances as a byproduct. Moreover, we show how to efficiently compute a rather tight bound for the time which is needed to solve a given instance. Finally, we prove that the problem of strong bisimilarity between processes of one-counter automata and finite-state processes is in **P**.

1 Introduction

In concurrency theory, *processes* are typically understood as (being associated with) states in *transition systems*, a fundamental and widely accepted model of discrete systems. Formally, a transition system is a triple $\mathcal{T} = (S, \Sigma, \rightarrow)$ where S is a set of *states*, Σ is a finite set of *actions* (or *labels*), and $\rightarrow \subseteq S \times \Sigma \times S$ is a *transition relation*. We write $s \xrightarrow{a} t$ instead of $(s, a, t) \in \rightarrow$ and we extend this notation to elements of Σ^* in the natural way. A state t is *reachable* from a state s iff there is $w \in \Sigma^*$ such that $s \xrightarrow{w} t$. A system $\mathcal{T}$ is *finite-state* iff the set of states of $\mathcal{T}$ is finite.

The *equivalence approach* to formal verification of concurrent systems is based on the following scheme: One describes the *specification* (the intended behaviour) $\mathcal{S}$ and the *implementation* $\mathcal{I}$ of a given system in some 'higher' formalism whose semantics is given in terms of transition systems, and then it is shown that $\mathcal{S}$ and $\mathcal{I}$ are *equivalent*. Actually, there are many ways how to capture the notion of process equivalence (see, e.g., [18]). It seems, however, that *bisimulation equivalence* [15,13] is of special importance, as its accompanying theory has been developed very intensively. Let $\mathcal{T} = (S, \Sigma, \rightarrow)$ be a transition system. A binary relation $R \subseteq S \times S$ is a *bisimulation* iff whenever $(s, t) \in R$, then for each $s \xrightarrow{a} s'$ there is some $t \xrightarrow{a} t'$ such that $(s', t') \in R$, and for each $t \xrightarrow{a} t'$ there is some $s \xrightarrow{a} s'$ such that $(s', t') \in R$. States s, t are *bisimulation equivalent* (or *bisimilar*), written $s \sim t$, iff there is a bisimulation relating them. Bisimulations can

* Supported by the Grant Agency of the Czech Republic, grants No. 201/98/P046 and No. 201/00/0400.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 317–328, 2000.
© Springer-Verlag Berlin Heidelberg 2000

also be used to relate states of *different* transition systems; formally, two systems can be considered as a single one by taking their disjoint union. An important variant of bisimilarity is *weak bisimilarity* introduced by Milner in his work on CCS [13]. This relation distinguishes between 'external' and 'internal' computational steps, and allows to 'ignore' the internal steps (which are usually denoted by a distinguished action τ) to a certain extent. Formally, we define the *extended transition relation* $\Rightarrow \subseteq S \times \Sigma \times S$ as follows: $s \overset{\tau}{\Rightarrow} t$ iff t is reachable from s via a finite (and possibly empty) sequence of transitions labelled by τ (note that $s \overset{\tau}{\Rightarrow} s$ for each s), and $s \overset{a}{\Rightarrow} t$ where $a \neq \tau$ iff there are states u, v such that $s \overset{\tau}{\Rightarrow} u \overset{a}{\rightarrow} v \overset{\tau}{\Rightarrow} t$. The relation of *weak bisimulation* is defined in the same way as bisimulation, but '$\Rightarrow$' is used instead of '$\rightarrow$'. Processes s, t are *weakly bisimilar*, written $s \approx t$, iff there is a weak bisimulation relating them. To prevent a confusion about bisimilarity and weak bisimilarity, we refer to bisimilarity as *strong bisimilarity* in the rest of this paper.

In this paper we study the complexity of checking strong and weak bisimilarity between processes of transition systems generated by (certain subclasses of) *pushdown automata* and processes of finite-state systems. A *pushdown automaton* is a tuple $\mathcal{P} = (Q, \Gamma, \Sigma, \delta)$ where Q is a finite set of *control states*, Γ is a finite *stack alphabet*, Σ is a finite *input alphabet*, and $\delta : (Q \times \Gamma) \rightarrow 2^{\Sigma \times (Q \times \Gamma^*)}$ is a *transition function* with finite image. We can assume (w.l.o.g.) that each transition increases the height (or length) of the stack at most by one (each PDA can be efficiently transformed to this kind of normal form). To $\mathcal{P}$ we associate the transition system $\mathcal{T}_{\mathcal{P}}$ where $Q \times \Sigma^*$ is the set of states, Σ is the set of actions, and the transition relation is determined by $(p, A\alpha) \overset{a}{\rightarrow} (q, \beta\alpha)$ iff $(a, (q, \beta)) \in \delta(p, A)$. As usual, we write $p\gamma$ instead of (p, γ) and we use ε to denote the empty word. The size of $\mathcal{P}$ is the length of a string which is obtained by writing all elements of the tuple linearly in binary. The size of a process $p\alpha$ of is the length of its corresponding binary encoding. Pushdown processes (i.e., processes of pushdown automata) have their origin in theory of formal languages [5], but recently (i.e., in the last decade) they have been found appropriate also in the context of concurrency theory because they provide a natural and important model of sequential systems. In this paper we mainly concentrate on a subclass of pushdown automata where the stack behaves like a *counter*. Such a restriction is reasonable because in practice we often meet systems which can be abstracted to finite-state programs operating on a single unbounded variable. For example, network protocols can maintain the count on how many unacknowledged messages have been sent, printer spool should know how many processes are waiting in the input queue, etc. Formally, a *one-counter automaton* $\mathcal{A}$ is a pushdown automaton with just two stack symbols I and Z; the transition function δ of $\mathcal{A}$ is a union of functions δ_Z and δ_I where $\delta_Z : (Q \times \{Z\}) \rightarrow 2^{\Sigma \times (Q \times (\{I\}^* \{Z\}))}$ and $\delta_I : (Q \times \{I\}) \rightarrow 2^{\Sigma \times (Q \times \{I\}^*)}$. Hence, Z works like a bottom symbol (which cannot be removed), and the number of I's which are stored in the stack represents the counter value. Processes of $\mathcal{A}$ (i.e., states of $\mathcal{T}_{\mathcal{A}}$) are of the form pI^iZ. In the rest of this paper we adopt a more intuitive notation, writing $p(i)$ instead of pI^iZ. It is worth to note that the size of $p(i)$ is $\mathcal{O}(i)$ and *not* $\mathcal{O}(\log i)$, because $p(i)$ is just a *symbolic* abbreviation for $p\alpha Z$ where α is a string of i symbols I. Again, we assume (w.l.o.g) that each transition increases the counter at most by one. A proper subclass of one-counter automata of its own interest are *one-counter nets*. Intuitively, OC-nets are 'weak' OC-

automata which cannot test for zero explicitly. They are computationally equivalent to a subclass of Petri nets [16] with (at most) one unbounded place. Formally, a *one-counter net* $\mathcal{N}$ is a one-counter automaton such that whenever $(a, qI^iZ) \in \delta_Z(p, Z)$, then $(a, qI^{i+1}) \in \delta_Z(p, I)$. In other words, each transition which is enabled at zero-level is also enabled at (each) non-zero-level. Hence, there are no 'zero-specific' transitions which could be used to 'test for zero'.

Observe that the out-going transitions of a OC process $q(i)$ where $i > 0$ do not depend on the actual value of i. Hence, the structure of transition systems which are associated with OC-automata (and, in particular, with OC-nets) is rather regular—they consist of a 'zero pattern' and a 'non-zero pattern' which is repeated infinitely often. Despite this regularity, some problems for OC-automata (and even for OC-nets) are computationally hard, as we shall see in the next section.

Now we give a short summary of relevant results for PDA and OC automata. The decidability of strong bisimilarity for processes of stateless PDA (which are also known as BPA processes) is due to [3]. Another (incomparable) positive result is [6] where it is shown that strong bisimilarity is decidable for processes of OC-automata. These results have been recently extended to general PDA in [17]. The problem of weak bisimilarity is still open for all of the mentioned (sub)classes. The decidability of strong/weak bisimilarity between processes of a (general) class $\mathcal{C}$ and finite-state ones has been studied in [7]. It is shown that the problem can be reduced to the model-checking problem for a temporal logic EF and processes of $\mathcal{C}$. Since EF is decidable for PDA processes, it suffices for showing the decidability, but the obtained algorithm is not very efficient—we only obtain **EXPTIME** upper-bound in this way for both strong and weak bisimilarity. Recently, **PSPACE** lower-bound for the problem of strong (and hence also weak) bisimilarity between PDA and FS processes has been given in [12]. A somewhat surprising result is [11] which says that strong and weak bisimilarity between BPA processes and finite-state ones is in **P**. OC-nets are studied, e.g., in [1,9] where it is shown that simulation equivalence (which is coarser than strong bisimilarity) is decidable for processes of OC-nets, and in [8] where a close relationship between simulation problems for OC-nets and the corresponding bisimulation problems for OC-automata is established.

In this paper we concentrate on the complexity of checking strong and weak bisimilarity between processes of OC-automata and FS processes. Our motivation is that the specification or the implementation of a system which is to be verified (see above) can often be specified as a finite-state process. Moreover, a number of 'classical' verification problems (e.g., liveness, safety) can be easily reduced to the problem of weak bisimilarity with a finite-state system. For example, if we want to check that the action a is *live* for a process g (i.e., each state which is reachable from g can reach a state which can emit a), we can rename all actions of g except a to τ and then check weak bisimilarity between g and f where f is a one-state process with the only transition $f \xrightarrow{a} f$.

In Section 2, it is shown that the problem of weak bisimilarity between processes of OC-nets and FS processes is **DP**-hard, even for a fixed finite-state process (intuitively, the class **DP** [14] is expected to be somewhat larger than the union of **NP** and **co-NP**; however, it is still contained in the $\Delta_2 = \mathbf{P}^{\mathbf{NP}}$ level of the polynomial hierarchy). Here we have to devise a special technique for encoding, guessing, and checking assignments of Boolean variables in the structure of OC-nets. As transition systems

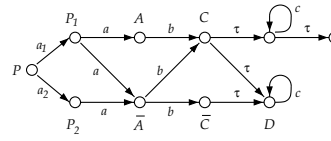
which are associated with OC-nets are rather regular, the method is not straightforward (observe that assignments are easy to handle with a stack; it is not so easy if there is only (one) counter at our disposal). Using the same technique we also show that strong bisimilarity between processes of OC-nets is **co-NP**-hard (strong bisimilarity between processes of OC-automata and finite-state processes is already *polynomial*—see below). Assuming the expected relationship among complexity classes, the **DP**-hardness result for weak bisimilarity actually says that any deterministic algorithm which decides the problem requires exponential time in the worst case. Rather than trying to establish **DP**-completeness, we turn our attention to a more 'practical' direction—in Section 3 we design an algorithm which decides weak bisimilarity between a process $p(i)$ of a OC-automaton $\mathcal{A}$ and a process f of a finite-state system $\mathcal{F}$ in time $\mathcal{O}(n^3\, m^5\, z^3\, (i+1))$ where n is the size of $\mathcal{A}$, m is the size of $\mathcal{F}$, and z is a special constant which depends on $\mathcal{A}$. So, if there was no z, or if z was always 'small', the problem would be in **P**. However, z *can* be much (exponentially) larger than n in general. However, it follows from the way how z is defined that the automaton must be *very* perverse to make its associated z large (a good example is the automaton constructed in the **DP**-hardness proof of Section 2). Hence, we conclude that our algorithm is actually efficient for many (if not all) practical instances, giving a sort of 'characterization' of all hard instances as a byproduct. Another advantage of our algorithm is that we can efficiently estimate the time which is needed to solve a given instance—although the computation of z for a given automaton $\mathcal{A}$ may take exponential time in general, we can efficiently (i.e., in polynomial time) compute a quite reliable bound for z. All hard instances are efficiently recognized in this way; it can also happen that some 'easy' instance is incorrectly declared as hard, but we argue that such situations are quite rare. The algorithm also works for strong bisimilarity, but in this case it only needs polynomial time—we obtain (as a simple consequence) that the problem of strong bisimilarity between OC processes and finite-state ones is in **P**. Proofs which were omitted due to space constraints can be found in [10].

2 Lower Bounds

In this section we show that the problem of weak bisimilarity between processes of OC-nets and finite-state processes is **DP**-hard (even for a *fixed* finite-state process), and that the problem of strong bisimilarity between processes of OC-nets is **co-NP**-hard.

Theorem 1. *The problem of weak bisimilarity between processes of one-counter nets and finite-state processes is* **DP***-hard.*

Proof. For purposes of this proof, we first fix the following finite-state system $\mathcal{F}$:



We show **DP**-hardness by reduction of the **DP**-complete problem SAT-UNSAT. An instance of the SAT-UNSAT problem is a pair (φ_1, φ_2) of Boolean formulae in CNF. The question is whether φ_1 is satisfiable and φ_2 unsatisfiable. First, we describe a

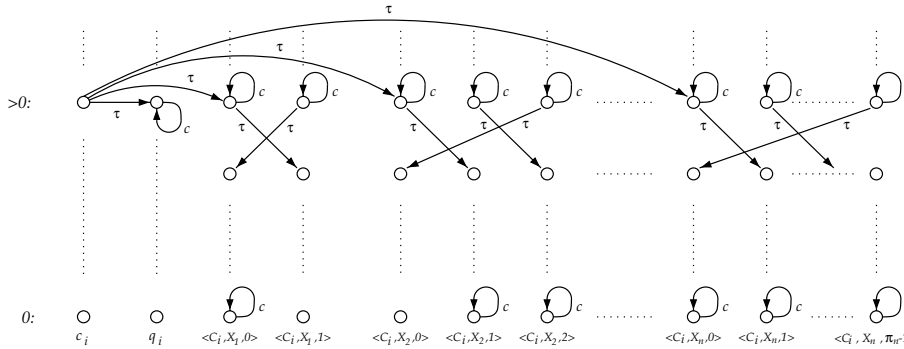
polynomial algorithm which for a given formula φ in CNF constructs a one-counter net $\mathcal{N}_\varphi$ and its process $s_\varphi(0)$ such that φ is satisfiable iff $s_\varphi(0) \approx P_1$, and φ is unsatisfiable iff $s_\varphi(0) \approx P_2$, where P_1, P_2 are the (fixed) FS processes of the system $\mathcal{F}$. It clearly suffices for our purposes, because then we can also construct a one-counter net $\mathcal{N}$ by taking the disjoint union of $\mathcal{N}_{\varphi_1}$, $\mathcal{N}_{\varphi_2}$ and adding a new control state s together with transitions $sZ \xrightarrow{a_1} s_{\varphi_1}Z, sI \xrightarrow{a_1} s_{\varphi_1}I$ and $sZ \xrightarrow{a_2} s_{\varphi_2}Z, sI \xrightarrow{a_2} s_{\varphi_2}I$ (the non-zero transitions are added just to fulfil the constraints of the definition of OC nets). Clearly (φ_1, φ_2) is a positive instance of the SAT-UNSAT problem iff $s(0) \approx P$ where P is the fixed FS process of the system $\mathcal{F}$.

In our proof we use the following theorem of number theory (see, e.g., [2]): Let π_i be the ith prime number, and let $f : \mathbb{N} \to \mathbb{N}$ be a function which assigns to each n the sum $\sum_{i=1}^{n} \pi_i$. Then f is $\mathcal{O}(n^3)$. (In our case, it suffices to know that the sum is asymptotically bounded by a polynomial in n.) With the help of this fact we can readily confirm that the below described construction is indeed polynomial.

Let $\varphi \equiv C_1 \wedge \cdots \wedge C_m$ be a formula in CNF where C_i are clauses over propositional variables $x_1, \cdots, x_n$. We assume (w.l.o.g.) that for every assignment $\nu : \{x_1, \cdots, x_n\} \to \{\texttt{true}, \texttt{false}\}$ there is at least one clause C_i such that $\nu(C_i) = \texttt{true}$ (this can be achieved, e.g., by adding the clause $(x_1 \vee \neg x_1)$ to φ). Furthermore, we also assume that φ is not a tautology, i.e., there is at least one assignment ν such that $\nu(\varphi) = \texttt{false}$ (it actually means that there is a clause where no variable appears both positively and negatively). The construction of $\mathcal{N}_\varphi = (Q, \{I, Z\}, \{a, b, c, \tau\}, \delta)$ will be described in a stepwise manner. The sets Q and δ are initially empty. For each clause C_i, $1 \le i \le m$, we do the following:

- We add new control states c_i and q_i to Q. Moreover, for each variable x_j and each k such that $0 \le k < \pi_j$ we add to Q a control state $\langle C_i, x_j, k\rangle$ (observe that for each C_i we add $\mathcal{O}(n^4)$ states in this way).
- We add to δ the transition $q_i I \xrightarrow{c} q_i I$.
- For each $1 \le j \le n$ we add to δ the transitions $c_i I \xrightarrow{\tau} \langle C_i, X_j, 0\rangle I$ and $c_i I \xrightarrow{\tau} qI$.
- For all j, k such that $1 \le j \le n$ and $0 \le k < \pi_j$ we add to δ the transition $\langle C_i, x_j, k\rangle I \xrightarrow{\tau} \langle C_i, x_j, (k+1) \bmod \pi_j\rangle \varepsilon$.
- For all j, k such that $1 \le j \le n$ and $0 \le k < \pi_j$ we add to δ the 'loop' $\langle C_i, x_j, k\rangle I \xrightarrow{c} \langle C_i, x_j, k\rangle I$.
- If a variable x_j does *not* appear positively in a clause C_i, then we add to δ the loop $\langle C_i, x_j, 0\rangle Z \xrightarrow{c} \langle C_i, x_j, 0\rangle Z$.
- If a variable x_j does not appear negatively in a clause C_i, then we add to δ the loops $\langle C_i, x_j, k\rangle Z \xrightarrow{c} \langle C_i, x_j, k\rangle Z$ for every $1 \le k < \pi_j$.

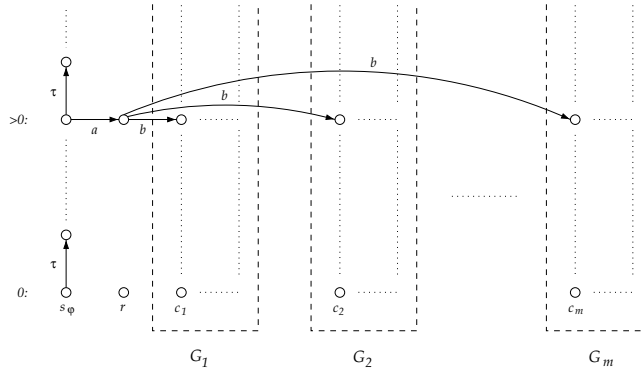
If we draw the transition system which is generated by the current approximation of $\mathcal{N}_\varphi$, we obtain a collection of G_i graphs, $1 \le i \le m$; each G_i corresponds to the 'subgraph' of $\mathcal{T}_\mathcal{N}$ which is obtained by restricting Q to the set of control states which have been added for the clause C_i. The structure of G_i is shown in the following picture. (To understand this figure, observe that transition systems associated to OC-automata can be viewed as two-dimensional 'tables' where column-indexes are control states and row-indexes are counter values $(0, 1, 2, \ldots)$. As the out-going transitions of a state $q(i)$ where $i > 0$ do not depend on the actual value of i, it suffices to depict the out-going transitions at zero and (some) non-zero level.)



Now we give several important facts about G_i (each fact easily follows from the previous ones):

- For each $l > 0$ we have that $c_i(l) \overset{\tau}{\Rightarrow} \langle C_i, x_j, k\rangle(0)$ iff $l \bmod \pi_j = k$.
- For each $l > 0$, the state $c_i(l)$ 'encodes' the (unique) assignment ν_l defined by $\nu_l(x_j) = \texttt{true}$ iff $c_i(l) \overset{\tau}{\Rightarrow} \langle C_i, x_j, 0\rangle(0)$; conversely, for each assignment ν there is $l \in \mathbb{N}$ such that $\nu = \nu_l$ (for example, we can put $l = \Pi_{j=0}^{n} f(j)$, where $f(j) = \pi_j$ if $\nu(x_j) = \texttt{true}$, and $f(j) = 1$ otherwise).
- For each $l > 0$ we have the following:
  - $\nu_l(C_i) = \texttt{false}$ iff $c_i(l) \approx \overline{C}$ for the state $\overline{C}$ of the system $\mathcal{F}$. Indeed, if $\nu_l(C_i) = \texttt{false}$, then $c_i(l)$ cannot reach any of the 'zero-states' where the action c is disabled—it can only emit c's (possibly with some intermediate τ's) without a possibility to terminate.
  - $\nu_l(C_i) = \texttt{true}$ (i.e., the clause C_i is true for ν_l) iff $c_i(l) \approx C$ for the state C of the system $\mathcal{F}$. This also explains the role of the control state q_i — we need it to make sure that the transition $C \overset{\tau}{\rightarrow} D$ can always be matched.

We finish the construction of $\mathcal{N}_\varphi$ by connecting the G_i components together. To do that, we add two new control states s_φ and r to Q, and enrich δ by adding the transitions $s_\varphi Z \overset{\tau}{\rightarrow} s_\varphi IZ$, $s_\varphi I \overset{\tau}{\rightarrow} s_\varphi I\,I$, $s_\varphi I \overset{\tau}{\rightarrow} rI$, and $rI \overset{b}{\rightarrow} c_i I$ for every $1 \leq i \leq m$. The structure of $\mathcal{T}_{\mathcal{N}_\varphi}$ is shown below.



Now we can observe the following:

- For each $l > 0$ we have that
  - $\nu_l(\varphi) = \texttt{true}$ iff $r(l) \approx A$ for the state A of the system $\mathcal{F}$. To see this, realize that $\nu_l(\varphi) = \texttt{true}$ iff $\nu_l(C_i) = \texttt{true}$ for each $1 \leq i \leq m$ iff $c_i(l) \approx C$ for each $1 \leq i \leq m$ due to the previous observations.
  - $\nu_l(\varphi) = \texttt{false}$ iff $r(l) \approx \overline{A}$ for the state $\overline{A}$ of the system $\mathcal{F}$. A proof is similar to the previous case; here we also need the assumption that at least one clause of φ is true for ν_l (so that we can be sure that the transition $\overline{A} \xrightarrow{b} C$ can be matched by $r(l)$).
- φ is unsatisfiable iff $s_\varphi(0) \approx P_2$ for the state P_2 of $\mathcal{F}$. Indeed, $s_\varphi(0)$ can perform its $\stackrel{a}{\Rightarrow}$ move only by going to some (arbitrary) $r(l)$. If φ is unsatisfiable, then $\nu_l(\varphi) = \texttt{false}$ for each such $r(l)$, hence all $\stackrel{a}{\Rightarrow}$ successors of $s_\varphi(0)$ are weakly bisimilar to $\overline{A}$ (see above), hence $s_\varphi(0) \approx P_2$. If φ is satisfiable, then there is a move $s_\varphi(0) \stackrel{a}{\Rightarrow} r(l)$ for some l such that $\nu_l(\varphi) = \texttt{true}$, hence $r(l) \approx A$ and $r(l) \not\approx \overline{A}$. Therefore, P_2 cannot match the $s_\varphi(0) \stackrel{a}{\Rightarrow} r(l)$ move and thus $s_\varphi(0) \not\approx P_2$.
- φ is satisfiable iff $s_\varphi(0) \approx P_1$ for the state P_1 of $\mathcal{F}$. It is checked in the same way as above. Here we use the assumption that φ is not a tautology, i.e., $s_\varphi(0)$ can always choose an assignment which makes φ false (i.e., $s_\varphi(0)$ can always match the transition $P_1 \xrightarrow{a} \overline{A}$). □

The main reason why we could not extend the hardness result to some higher complexity class (e.g., **PSPACE**) is that there is no apparent way how to implement a 'stepwise-guessing' of Boolean variables which would allow to encode, e.g., the QBF problem; each such attempt resulted in an exponential blow-up in the number of control states. However, we can still re-use our technique to prove the following:

Theorem 2. *The problem of strong bisimilarity between processes of one-counter nets is* **co-NP**-*hard.*

Proof. We use a similar construction as in the proof of Theorem 1. Given a formula φ in CNF, we construct two one-counter nets $\mathcal{N}, \overline{\mathcal{N}}$ and their processes $s(0), \overline{s}(0)$ such that φ is unsatisfiable iff $s(0) \sim \overline{s}(0)$. The net $\mathcal{N}$ is just a slight modification of the net $\mathcal{N}_\varphi$ of Theorem 1 — we only rename all τ-labels to c. A key observation is that φ is unsatisfiable iff after each sequence of transitions of the form c^*a (i.e., after each choice of an assignment) there is a b-transition to a state which can only emit an infinite sequence of c actions without a possibility to terminate (i.e., at least one clause is false for any assignment). The net $\overline{\mathcal{N}}$ is a 'copy' of $\mathcal{N}$ but we also add a new control state q and transitions $qI \xrightarrow{c} qI, \overline{r}I \xrightarrow{b} qI$ where $\overline{r}$ is a 'twin' of the state r of $\mathcal{N}_\varphi$. We put s and $\overline{s}$ to be the corresponding twins of the state s_φ of $\mathcal{N}_\varphi$. Now we can easily check that φ is unsatisfiable iff $s(0) \sim \overline{s}(0)$ — the crucial argument is stated above. □

3 Efficient Algorithms

In this section we design an algorithm which decides weak bisimilarity between processes of OC-automata and finite-state processes. As expected, the algorithm requires

exponential time in the worst case. However, it works rather efficiently for many (and we believe that almost all) 'practical' instances. It also works for strong bisimilarity where it needs only polynomial time.

Let $\mathcal{T} = (S, \Sigma, \rightarrow)$ be a transition system. For each $i \in \mathbb{N}_0$ we define the relation $\approx_i$ inductively as follows: $\approx_0 = S \times S$; and $s \approx_{i+1} t$ iff $s \approx_i t$ and for each $s \stackrel{a}{\Rightarrow} s'$ there is some $t \stackrel{a}{\Rightarrow} t'$ such that $s' \approx_i t'$, and vice versa. (The $\approx_i$ relations are also used to relate states of different transition systems; formally, we consider two transition systems to be a single one by taking their disjoint union.) Our algorithm relies on the following theorem established in [7]:

Theorem 3. *Let $\mathcal{G} = (G, \Sigma, \rightarrow)$ be a (general) transition system and $\mathcal{F} = (F, \Sigma, \rightarrow)$ a finite-state system. We say that a state $g \in G$ is* good *w.r.t. $i \in \mathbb{N}_0$ iff there is $f \in F$ such that $g \approx_i f$; g is* bad *w.r.t. i iff g is not good w.r.t. i.*

Let $g \in G$, $f \in F$, and $k = |F|$. It holds that $g \approx f$ iff $g \approx_k f$ and each state which is reachable from g is good w.r.t. k.

For the rest of this section we fix a one-counter automaton $\mathcal{A} = (Q, \{I, Z\}, \Sigma, \delta)$ of size n, and a finite-state system $\mathcal{F} = (F, \Sigma, \rightarrow)$ of size m.

To decide weak bisimilarity between processes $p(i)$ of $\mathcal{A}$ and f of $\mathcal{F}$, it suffices (by Theorem 3) to find out if $p(i) \approx_m f$ and whether $p(i)$ can reach a state which is bad w.r.t. m. We do that by constructing a constant z such that for each state $q(j)$ of $\mathcal{T}_{\mathcal{A}}$ where $j \geq (4m+1)z$ we have that $q(j) \approx_m q(j-z)$. In other words, each state of $\mathcal{T}_{\mathcal{A}}$ is (up to $\approx_m$) represented by another (and effectively constructible) state whose counter value is bounded by $(4m+1)z$. Then we convert this 'initial part' of $\mathcal{T}_{\mathcal{A}}$ to a finite-state system $\mathcal{F}_{\mathcal{A}}$ and construct the $\approx_m$ relation between states of $\mathcal{F}_{\mathcal{A}}$ and $\mathcal{F}$. The question if $p(i) \approx_m f$ is then easy to answer (we look if the representant of $p(i)$ within $\mathcal{F}_{\mathcal{A}}$ is related with f by $\approx_m$). The question if $p(i)$ can reach a state which is bad w.r.t. m still requires some development—we observe that states which are bad w.r.t. m are 'regularly distributed' in $\mathcal{T}_{\mathcal{A}}$ and construct a description of that distribution (which is 'read' from $\mathcal{F}_{\mathcal{A}}$) in a form of a finite-state automaton $\mathcal{M}$ which recognizes all bad states. Then we use an algorithm of [4] which constructs from $\mathcal{M}$ an automaton $\mathcal{M}'$ recognizing the set of all states which can reach a state recognized by $\mathcal{M}$, and look whether $\mathcal{M}'$ accepts $p(i)$. All procedures we use are polynomial in the size of $\mathcal{F}_{\mathcal{A}}$. Hence, it is only the size of z which can make the problem computationally hard. The construction of z can require exponential time; however, we give an algorithm which efficiently (i.e., in polynomial time) computes a reliable upper bound Z for z.

Intuitively, the only difference between processes $p(i), p(j)$, where $i \neq j$, is the way how they can access the 'zero level'. As long as the counter remains positive, each process can 'mimic' moves of the other process by entering the same control state and performing the same operation on the counter. Observe that the counter can be generally decremented by an unbounded value in a single $\stackrel{a}{\Rightarrow}$ step (due to an unbounded number of τ-transitions). The next definitions and lemmata reveal a crucial periodicity in the structure of $\mathcal{T}_{\mathcal{A}}$ which shows that decrementing the counter 'too much' in one $\stackrel{a}{\Rightarrow}$ step is not the thing which allows to demonstrate a possible difference between $p(i), p(j)$.

For each $l \in \mathbb{N}_0$ we define a family of binary relations $\stackrel{a}{\Rightarrow}_l$, $a \in \Sigma$, over the set of states of $\mathcal{T}_{\mathcal{A}}$ as follows: $p(i) \stackrel{a}{\Rightarrow}_l q(j)$ iff there is a sequence of transitions from $p(i)$ to

$q(j)$ which forms one '$\overset{a}{\Rightarrow}$' move and the counter value remains greater or equal l in all states which appear in the sequence (including $p(i)$ and $q(j)$).

Definition 1. *We define a function* $step_{\mathcal{A}} : Q \to 2^Q$ *by* $step_{\mathcal{A}}(p) = \{q \in Q \mid p(2) \overset{\tau}{\Rightarrow}_1 q(1)\}$.

Since the reachability problem for one-counter automata (and even for pushdown automata) is in **P**, the function $step_{\mathcal{A}}$ is effectively constructible in polynomial time. As the out-going transitions of a state $p(i)$ for $i > 0$ do not depend on the actual value of i, for each $i \in \mathbb{N}$ we have that $q \in step_{\mathcal{A}}(p)$ iff $p(i+1) \overset{\tau}{\Rightarrow}_i q(i)$.

We extend $step_{\mathcal{A}}$ to subsets of Q by $step_{\mathcal{A}}(M) = \bigcup_{p \in M} step_{\mathcal{A}}(p)$. For each $p \in Q$ we now define the sequence C_p inductively as follows: $C_p(1) = \{p\}$ and $C_p(i+1) = step_{\mathcal{A}}(C_p(i))$. The next lemma is easy to prove by a straightforward induction on i.

Lemma 1. *For all* $p \in Q$ *and* $i, j \in \mathbb{N}$ *we have that* $q \in C_p(j)$ *iff* $p(i+j) \overset{\tau}{\Rightarrow}_i q(i)$.

Another simple observation is that the sequence C_p is (for every $p \in Q$) of the form $C_p = \alpha_p \beta_p^\omega$ where α_p, β_p are finite sequences of pairwise different subsets of Q (due to the assumption that the elements of α_p and β_p are pairwise different we also have that α_p and β_p are *unique*). Note that β_p can also consist of just one element $\emptyset$. We define the *prefix* and *period* of p, denoted $pre(p)$ and $per(p)$, to be the length of α_p and β_p, respectively. Now we put

$$z = \max\{pre(p) \mid p \in Q\} \cdot \mathrm{lcm}\{per(p) \mid p \in Q\}$$

where $\mathrm{lcm}(M)$ denotes the least common multiply of elements of M. As we shall see, $\max\{pre(p) \mid p \in Q\}$ is always $\mathcal{O}(n^2)$. However, $\mathrm{lcm}\{per(p) \mid p \in Q\}$ can be *exponential* in n (for example, examine the net $\mathcal{N}_\varphi$ constructed in the proof of Theorem 1). As we already mentioned, the size of z is the only thing which can make the considered problem hard. Hence, we obtain a kind of 'characterization' of all hard instances—OC-automata which are presented in hard instances must contain many 'decreasing τ-cycles' of an incomparable length. Also observe that the construction of z can require exponential time, because $per(p)$ for a given p can be exponential in n (in the end of this section we show how to compute a reasonable upper bound Z for z efficiently). The following lemma is immediate:

Lemma 2. *For all* $p \in Q$ *and* $i \geq z$ *we have that* $C_p(i) = C_p(i+z)$.

The next three lemmata provide a crucial observation about the structure of $\mathcal{T}_{\mathcal{A}}$ and precisely formulate the intuition that 'decreasing the counter too much in one $\overset{a}{\Rightarrow}$ step does not help'. Proofs can be found in [10].

Lemma 3. *For all* $p \in Q$ *and* $j \in \mathbb{N}$ *it holds that*

- *if there is a sequence of* τ*-transitions from* $p(j+2z)$ *to (some)* $q(l)$ *which decreases the counter to* j *at some point, then* $p(j+z) \overset{\tau}{\Rightarrow} q(l)$*;*
- *if there is a sequence of* τ*-transitions from* $p(j+z)$ *to (some)* $q(l)$ *which decreases the counter to* j *at some point, then* $p(j+2z) \overset{\tau}{\Rightarrow} q(l)$*;*

Lemma 4. *For all $p \in Q$ and $j \in \mathbb{N}$ it holds that*

- *if there is a sequence of transitions forming one '$\overset{a}{\Rightarrow}$' move from $p(j+4z)$ to (some) $q(l)$ which decreases the counter to j at some point, then $p(j+3z) \overset{a}{\Rightarrow} q(l)$;*
- *if there is a sequence of transitions forming one '$\overset{a}{\Rightarrow}$' move from $p(j+3z)$ to (some) $q(l)$ which decreases the counter to j at some point, then $p(j+4z) \overset{a}{\Rightarrow} q(l)$;*

Lemma 5. *Let $p \in Q$ and $k \in \mathbb{N}_0$. For each $c > (4k+1)z$ we have that $p(c) \approx_k p(c-z)$.*

Now we are almost in a position to prove the first main theorem of this section. It remains to extend our equipment with the following tool:

Definition 2. *Let $\mathcal{P} = (Q, \Gamma, \Sigma, \delta)$ be a pushdown automaton, $\mathcal{M} = (S, \Gamma, \gamma, F)$ a nondeterministic finite-state automaton (note that the input alphabet of $\mathcal{M}$ is the stack alphabet of $\mathcal{P}$), and $Init : Q \to S$ a total function. A process $p\alpha$ of $\mathcal{P}$ is* recognized *by the pair $(\mathcal{M}, Init)$ iff $\hat{\gamma}(Init(p), \alpha) \cap F \neq \emptyset$ where $\hat{\gamma}$ is the natural extension of γ to elements of $S \times \Gamma^*$.*

The next theorem is taken from [4].

Theorem 4. *Let $\mathcal{P} = (Q, \Gamma, \Sigma, \delta)$ be a pushdown automaton, $\mathcal{M} = (S, \Gamma, \gamma, F)$ a finite-state automaton, and $Init : Q \to S$ a total function. Let N be the set of processes recognized by $(\mathcal{M}, Init)$. Then one can effectively construct an automaton $\mathcal{M}' = (S, \Gamma, \gamma', F)$ in time $\mathcal{O}(|\delta| \cdot |S|^3)$ such that $(\mathcal{M}', Init)$ recognizes the set*

$$Pre^*(N) = \{q\beta \mid q\beta \to^* p\alpha \text{ for some } p\alpha \in N\}$$

of all predecessors of N.

Theorem 5. *The problem of weak bisimilarity between processes $p(i)$ of $\mathcal{A}$ and f of $\mathcal{F}$ is decidable in $\mathcal{O}(n^3\, m^5\, z^3\, (i+1))$ time.*[1]

Proof. By Theorem 3, we need to find out whether $p(i) \approx_m f$ and whether $p(i)$ can reach a state which is bad w.r.t. m. Due to Lemma 5 we know that the set of all states of $\mathcal{T}_{\mathcal{A}}$ up to $\approx_m$ can be represented by the subset of states of $\mathcal{T}_{\mathcal{A}}$ where the counter value is at most $(4m+1)z$. Formally, we first define the function $\mathcal{B} : (Q \times \mathbb{N}_0) \to (Q \times \mathbb{N}_0)$ as follows (where $\langle q, j\rangle$ is just another notation for $q(j)$):

$$\mathcal{B}(\langle q, j\rangle) = \begin{cases} \langle q, j\rangle & \text{if } j \leq (4m+1)z; \\ \langle q, (4m+1)z\rangle & \text{if } j > (4m+1)z \text{ and } (j \bmod z) = 0; \\ \langle q, 4mz + (j \bmod z)\rangle & \text{if } j > (4m+1)z \text{ and } (j \bmod z) \neq 0. \end{cases}$$

An immediate consequence of Lemma 5 is that for all $q \in Q$ and $j \in \mathbb{N}_0$ we have $q(j) \approx_m \mathcal{B}(q(j))$. Now we define a finite-state system $\mathcal{F}_{\mathcal{A}} = (F_{\mathcal{A}}, \Sigma, \hookrightarrow)$ where $F_{\mathcal{A}}$ is the image of $\mathcal{B}$ (i.e., $F_{\mathcal{A}} = \{q(j) \mid q \in Q, 0 \leq j \leq (4m+1)z\}$), Σ is the set of actions

[1] Note that we need a non-constant time even in the particular case when $i = 0$ (the problem is still **DP**-hard). That is why we write '$i+1$'.

of $\mathcal{A}$, and $\hookrightarrow$ is the least relation satisfying the following: if $r(k) \xrightarrow{a} s(l)$ is a transition of $\mathcal{T}_\mathcal{A}$, then $\mathcal{B}(r(k)) \overset{a}{\hookrightarrow} \mathcal{B}(s(l))$. Observe that $\mathcal{F}_\mathcal{A}$ is actually the 'initial part' of $\mathcal{T}_\mathcal{A}$; the only difference is that all up-going transitions of states at level $(4m+1)z$ are 'bent' down to the corresponding $\approx_m$-equivalent states at level $4mz+1$. Note that for each $q(j)$ we still have that $q(j) \approx_m \mathcal{B}(q(j))$ (when $\mathcal{B}(q(j))$ is seen as a state of $\mathcal{F}_\mathcal{A}$). The number of states of $\mathcal{F}_\mathcal{A}$ is $\mathcal{O}(n\,m\,z)$; moreover, the number of out-going transitions at each 'level' of $\mathcal{T}_\mathcal{A}$ is $\mathcal{O}(n)$, hence the size of $\hookrightarrow$ is $\mathcal{O}(n\,m\,z)$, which means that the total size of $\mathcal{F}_\mathcal{A}$ is also $\mathcal{O}(n\,m\,z)$.

Now, let us realize that if we have a finite-state system of size t, it takes $\mathcal{O}(t^3)$ time to compute the associated '$\overset{a}{\Rightarrow}$' relation (for each state s and action a we need $\mathcal{O}(t)$ time to compute the set $\{r \mid s \overset{a}{\Rightarrow} r\}$). Therefore, we need $\mathcal{O}(n^3\,m^3\,z^3)$ time to construct the extended transition relations for $\mathcal{F}_\mathcal{A}$ and $\mathcal{F}$. To compute the $\approx_m$ relation between the states of $\mathcal{F}_A$ and $\mathcal{F}$, we define $\mathcal{R}^0 = F_\mathcal{A} \times F$, and $\mathcal{R}^{i+1} = \mathit{Exp}(\mathcal{R}^i)$ where the function $\mathit{Exp} : (F_\mathcal{A} \times F) \to (F_\mathcal{A} \times F)$ refines its argument according to the definition of $\approx_i$ — a pair $(r(j), g)$ belongs to $\mathit{Exp}(\mathcal{R})$ iff it belongs to $\mathcal{R}$ and for each '$\overset{a}{\Rightarrow}$' move of one component there is a corresponding '$\overset{a}{\Rightarrow}$' move of the other component such that the resulting pair of states belongs to $\mathcal{R}$. Clearly, for each pair $(r(j), g)$ of $F_\mathcal{A} \times F$ we have that $r(j) \approx_m g$ iff $(r(j), g) \in \mathcal{R}^m$. It remains to clarify the time costs. The function Exp is computed m times. Each time, $\mathcal{O}(n\,m^2\,z)$ pairs are examined. For each such pair we have to check the membership to $\mathit{Exp}(\mathcal{R})$. This takes only $\mathcal{O}(n\,m^2\,z)$ time, because the extended transition relations have already been computed. To sum up, we need $\mathcal{O}(n^3\,m^5\,z^3)$ time in total.

To check if $p(i) \approx_m f$, we simply look if $(\mathcal{B}(p(i), f)) \in \mathcal{R}^m$. It remains to find out whether $p(i)$ can reach a state $q(j)$ which is bad w.r.t. m. Observe that $q(j)$ is bad w.r.t. m iff the state $\mathcal{B}(q(j))$ of $\mathcal{F}_\mathcal{A}$ is bad w.r.t. m. Therefore, we can easily construct a finite-state automaton $\mathcal{M}$ and a function Init such that the pair $(\mathcal{M}, \mathit{Init})$ recognizes the set of all bad states of $\mathcal{T}_\mathcal{A}$ — we put $\mathcal{M} = (S, \{I, Z\}, \gamma, \{\mathit{fin}\})$ where $S = \{\mathit{fin}\} \cup \{p(i) \mid p \in Q, 0 \le i \le (4m+1)z\}$ and γ is the least transition function satisfying the following:

- $p(i+1) \in \gamma(p(i), I)$ for all $p \in Q$, $0 \le i < (4m+1)z$;
- $p(4mz+1) \in \gamma(p((4m+1)z), I)$ for each $p \in Q$;
- if a state $p(i)$ of $\mathcal{F}_\mathcal{A}$ is bad, then $\mathit{fin} \in \gamma(p(i), Z)$.

The function Init is defined by $\mathit{Init}(p) = p(0)$ for all $p \in Q$. Note that $\mathcal{M}$ has $\mathcal{O}(n\,m\,z)$ states. Now we compute the automaton $\mathcal{M}'$ of Theorem 4 (it takes $\mathcal{O}(n^2\,m\,z)$ time) and check if $(\mathcal{M}', \mathit{Init})$ recognizes $p(i)$. This can be done in $\mathcal{O}(n\,m\,z\,(i+1))$ time because $\mathcal{M}'$ has the same set of states as $\mathcal{M}$.

We see that $\mathcal{O}(n^3\,m^5\,z^3\,(i+1))$ time suffices for all of the aforementioned procedures. □

Our algorithm also works for strong bisimilarity in the following way: If we are to decide strong bisimilarity between $p(i)$ and f, we first rename all τ-transitions of $\mathcal{A}$ and $\mathcal{F}$ with some (fresh) action e (it does not change anything from the point of view of strong bisimilarity, because here the τ-transitions are treated as 'ordinary' ones). As there are no τ-transitions anymore, there is no difference between strong and weak bisimilarity, hence we can use the designed algorithm. Also observe that if there are no τ's then $z = 1$, so we can conclude:

Corollary 1. *The problem of strong bisimilarity between processes $p(i)$ of $\mathcal{A}$ and f of $\mathcal{F}$ is in* **P**.

As we already mentioned, the construction of z can take exponential time. Now we show how to compute a rather tight upper bound Z for z in polynomial time. The associated lemmata and proofs can be found in [10].

Theorem 6. *We say that $p \in Q$ is* self-embedding *iff $p \in C_p(i)$ for some $i \geq 2$. Let us define $Z = (|Q|^2 + |Q|) \cdot \mathrm{lcm}\{per(p) \mid p \in Q$ is self-embedding $\}$. Then Z can be computed in time which is polynomial in n. Moreover, $z \leq Z$.*

References

1. P.A. Abdulla and K. Čerāns. Simulation is decidable for one-counter nets. In *Proceedings of CONCUR'98*, volume 1466 of *LNCS*, pages 253–268. Springer, 1998.
2. E. Bach and J. Shallit. *Algorithmic Number Theory. Vol. 1, Efficient Algorithms*. The MIT Press, 1996.
3. S. Christensen, H. Hüttel, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 121:143–148, 1995.
4. J. Esparza and P. Rossmanith. An automata approach to some problems on context-free grammars. In *Foundations of Computer Science, Potential – Theory – Cognition*, volume 1337 of *LNCS*, pages 143–152. Springer, 1997.
5. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
6. P. Jančar. Bisimulation equivalence is decidable for one-counter processes. In *Proceedings of ICALP'97*, volume 1256 of *LNCS*, pages 549–559. Springer, 1997.
7. P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proceedings of ICALP'98*, volume 1443 of *LNCS*, pages 200–211. Springer, 1998.
8. P. Jančar, A. Kučera, and F. Moller. Simulation and bisimulation over one-counter processes. In *Proceedings of STACS 2000*, volume 1770 of *LNCS*, pages 334–345. Springer, 2000.
9. P. Jančar, F. Moller, and Z. Sawa. Simulation problems for one-counter machines. In *Proceedings of SOFSEM'99*, volume 1725 of *LNCS*, pages 404–413. Springer, 1999.
10. A. Kučera. Efficient verification algorithms for one-counter processes. Technical report FIMU-RS-2000-03, Faculty of Informatics, Masaryk University, 2000.
11. A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proceedings of CONCUR'99*, volume 1664 of *LNCS*, pages 368–382. Springer, 1999.
12. R. Mayr. *Private communication*. 1999.
13. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
14. Ch. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
15. D.M.R. Park. Concurrency and automata on infinite sequences. In *Proceedings 5th GI Conference*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.
16. W. Reisig. *Petri Nets—An Introduction*. Springer, 1985.
17. G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite outdegree. In *Proceedings of 39th Annual Symposium on Foundations of Computer Science*, pages 120–129. IEEE Computer Society Press, 1998.
18. R.J. van Glabbeek. The linear time—branching time spectrum. In *Proceedings of CONCUR'90*, volume 458 of *LNCS*, pages 278–297. Springer, 1990.

On the Complexity of Bisimulation Problems for Basic Parallel Processes

Richard Mayr

LIAFA - Université Denis Diderot - Case 7014 - 2, place Jussieu,
F-75251 Paris Cedex 05. France.
mayr@liafa.jussieu.fr

Abstract. Strong bisimilarity of Basic Parallel Processes (BPP) is decidable, but the best known algorithm has non-elementary complexity [7]. On the other hand, no lower bound for the problem was known. We show that strong bisimilarity of BPP is co-$\mathcal{NP}$-hard.
Weak bisimilarity of BPP is not known to be decidable, but an $\mathcal{NP}$ lower bound has been shown in [31]. We improve this result by showing that weak bisimilarity of BPP is Π_2^p-hard.
Finally, we show that the problems if a BPP is regular (i.e., finite) w.r.t. strong and weak bisimilarity are co-$\mathcal{NP}$-hard and Π_2^p-hard, respectively.

1 Introduction

Bisimulation equivalence plays a central role in the theory of process algebras [25]. The decidability and complexity of bisimulation problems for infinite-state systems has been studied intensively (see [26] for a survey). While many algorithms for bisimulation problems have a very high complexity, only few lower bounds are known.

The state of the art. Strong bisimilarity of two Petri nets and weak bisimilarity of a Petri net and a finite automaton is undecidable [13, 14]. Weak bisimilarity for Basic Parallel Processes (BPP) is $\mathcal{NP}$-hard and weak bisimilarity for context-free processes (BPA) is $PSPACE$-hard [31]. However, it is still an open question whether these problems are decidable.

Some lower bounds for decidable bisimulation problems have been shown in [23]. Strong (and weak) bisimilarity between pushdown automata (PDA) and finite automata is $PSPACE$-hard, finiteness of PDA w.r.t. weak and strong bisimilarity also $PSPACE$-hard. Finally, both strong bisimilarity of Petri nets and finite automata and finiteness of Petri nets w.r.t. strong bisimilarity are $EXPSPACE$-hard. (See the table in Section 5 for a summary of all results on the complexity of bisimulation problems.)

Basic Parallel Processes (BPP) were introduced by Christensen [6] as the fragment of CCS [25] without communication, restriction and relabeling. They are equivalent to communication-free nets [8], the subclass of Petri nets [28] where every transition has exactly one input-place with arc-weight one. While strong (and weak) bisimilarity are undecidable for Petri nets [13], strong bisimilarity is decidable for BPP (i.e., communication-free nets) [7]. However, the algorithm in [7] has non-elementary complexity and, to the best of our knowledge, no better algorithm has been found since then. In spite of this, no lower bound for the problem has been found either.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 329–341, 2000.
© Springer-Verlag Berlin Heidelberg 2000

However, there is a polynomial algorithm for bisimilarity on the restricted subclass of *normed* BPP [12]. (A process is *normed* iff from every reachable state there is a terminating computation.) Thus, it was conjectured that a polynomial algorithm should also exist for general (unnormed) BPP. This belief was reinforced by the fact that many other problems for BPP are polynomial: boundedness [17], termination, liveness, (partial) deadlock reachability and (partial) livelock reachability [21, 22]. (On the other hand there are also hard problems for BPP: reachability is $\mathcal{NP}$-complete [8], some model checking problems are $PSPACE$-complete [20, 22] or even undecidable [9].)

Our contribution. We show that strong bisimilarity for BPP is co-$\mathcal{NP}$-hard (thus proving the above mentioned conjecture wrong). We also show that weak bisimilarity for BPP is Π_2^p-hard, thus improving a previously established $\mathcal{NP}$ lower bound [31]. Finally, we show that the problem if a BPP is regular (i.e., finite) w.r.t. strong and weak bisimilarity is co-$\mathcal{NP}$-hard and Π_2^p-hard, respectively.

2 Definitions

Let $Act = \{a, b, c, \ldots\}$ and $Const = \{\epsilon, X, Y, Z, \ldots\}$ be disjoint countably infinite sets of *actions* and *process constants*, respectively. The class of *general process expressions* G is defined by $E \ ::= \ \epsilon \mid X \mid E \| E \mid E.E$, where $X \in Const$ and ϵ is a special constant that denotes the empty expression. Intuitively, '.' is a sequential composition and '$\|$' is a parallel composition. We do not distinguish between expressions related by *structural congruence* which is given by the following laws: '.' and '$\|$' are associative, '$\|$' is commutative, and 'ϵ' is a unit for '.' and '$\|$'.

A *process rewrite system* (PRS) [24] is specified by a finite set Δ of *rules* which have the form $E \xrightarrow{a} F$, where $E, F \in G$, $E \neq \epsilon$ and $a \in Act$. $Const(\Delta)$ and $Act(\Delta)$ denote the sets of process constants and actions which are used in the rules of Δ, respectively (note that these sets are finite). Each process rewrite system Δ defines a unique transition system where states are process expressions over $Const(\Delta)$. $Act(\Delta)$ is the set of labels. The transitions are determined by Δ and the following inference rules (remember that '$\|$' is commutative):

$$\frac{(E \xrightarrow{a} F) \in \Delta}{E \xrightarrow{a} F} \qquad \frac{E \xrightarrow{a} E'}{E.F \xrightarrow{a} E'.F} \qquad \frac{E \xrightarrow{a} E'}{E \| F \xrightarrow{a} E' \| F}$$

We extend the notation $E \xrightarrow{a} F$ to elements of Act^* in a standard way. Moreover, we say that F is *reachable* from E if $E \xrightarrow{w} F$ for some $w \in Act^*$.

Various subclasses of process rewrite systems can be obtained by imposing certain restrictions on the form of rules. To specify those restrictions, we first define the classes S and P of *sequential* and *parallel* expressions, composed of all process expressions which do not contain the '$\|$' and the '.' operator, respectively. We also use '$\mathsf{1}$' to denote the set of process constants. The hierarchy of process rewrite systems is presented in Fig. 1; the restrictions are specified by a pair (A, B), where A and B are the classes of expressions which can appear on the left-hand and the right-hand side of rules, respectively. This hierarchy contains almost all classes of infinite state systems which have been studied so far; BPA (Basic Process Algebra, also called context-free processes), BPP (Basic Parallel Processes), and PA-processes are well-known [1], PDA correspond to pushdown

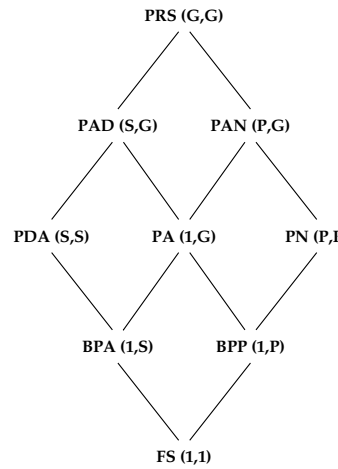


Fig. 1. A hierarchy of PRS

automata (as proved by Caucal in [5]), PN correspond to Petri nets, PRS stands for 'Process Rewrite Systems', PAD and PAN are artificial names made by combining existing ones (PAD = PA+PDA, PAN = PA+PN).

Here we study Basic Parallel Processes (BPP) that correspond to process rewrite systems of type $(\mathsf{1}, P)$.

We consider the semantical equivalences *weak bisimilarity* and *strong bisimilarity* [25] over transition systems generated by PRS.

Definition 1. *The action τ is a special 'silent' internal action. The* extended transition relation '$\stackrel{a}{\Rightarrow}$' *is defined by $E \stackrel{a}{\Rightarrow} F$ iff either $E = F$ and $a = \tau$, or $E \stackrel{\tau^i}{\rightarrow} E' \stackrel{a}{\rightarrow} E'' \stackrel{\tau^j}{\rightarrow} F$ for some $i, j \in \mathbb{N}_0$, $E', E'' \in G$. A binary relation R over process expressions is a* weak bisimulation *iff whenever $(E, F) \in R$ then for every $a \in Act$: if $E \stackrel{a}{\rightarrow} E'$ then there is $F \stackrel{a}{\Rightarrow} F'$ s.t. $(E', F') \in R$ and if $F \stackrel{a}{\rightarrow} F'$ then there is $E \stackrel{a}{\Rightarrow} E'$ s.t. $(E', F') \in R$. Processes E, F are* weakly bisimilar*, written $E \approx F$, iff there is a weak bisimulation relating them. Strong bisimulation is defined similarly with $\stackrel{a}{\rightarrow}$ instead of $\stackrel{a}{\Rightarrow}$. Processes E, F are* strongly bisimilar*, written $E \sim F$, iff there is a strong bisimulation relating them. The largest (strong or weak) bisimulation is an equivalence relation.*

Bisimulation equivalence can also be described by *bisimulation games* [30, 32] between two players. One player, the 'attacker', tries to prove that two given processes are not bisimilar, while the other player, the 'defender', tries to frustrate this. In every round of the game the attacker chooses one process and performs an action. The defender must imitate this move and perform the same action in the other process (possibly together with several internal τ-actions in the case of weak bisimulation). If one player cannot move then the other player wins. The defender wins every infinite game. Two processes are bisimilar iff the defender has a winning strategy and non-bisimilar iff the attacker has a winning strategy.

3 Hardness of Strong Bisimilarity for BPP

STRONG BISIMILARITY OF BPP

Instance: Two BPP processes P_1 and P_2.
Question: $P_1 \sim P_2$?

This problem has been shown to be decidable in [7]. However, the algorithm relies on Dickson's Lemma for termination and therefore the algorithm is not primitive recursive. A polynomial algorithm for bisimilarity on the restricted subclass of *normed* BPP has been described in [12], which led to the conjecture that the general problem was also polynomial. We prove this conjecture wrong by proving a co-$\mathcal{NP}$ lower bound. Thus no polynomial algorithm for strong bisimilarity of BPP can exist, unless $\mathcal{P} = \mathcal{NP}$.

First we give an intuition why the general (unnormed) problem is so hard, using the terminology of communication-free Petri nets. The problem if a place in a communication-free net is unbounded (i.e., if there are reachable states that put arbitrarily high numbers of tokens on it) is easily decidable in polynomial time [17]. However, it is not so easy to determine if the number of tokens on a place really matters w.r.t. bisimilarity, i.e., if states with different numbers of tokens on this place are really different w.r.t. bisimilarity (i.e., non-bisimilar). First we consider the simple example Δ: $X \xrightarrow{a} X\|Y,\ X \xrightarrow{a} \epsilon,\ Y \xrightarrow{a} \epsilon,\ Z \xrightarrow{a} Z$. The process (X, Δ) is infinite w.r.t. bisimilarity (since it has infinitely many non-bisimilar reachable states). However, $(X\|Z, \Delta)$ is finite w.r.t. bisimilarity, since $(X\|Z, \Delta) \sim (Z, \Delta)$. We say that in the process $(X\|Z, \Delta)$ the subprocess (Z, Δ) *masks* the infiniteness of (X, Δ). In particular, the subprocess Z has the effect that the number of subprocesses Y doesn't matter for bisimilarity, since $(Y^n\|Z, \Delta) \sim (Y^m\|Z, \Delta)$ for any $n, m \in \mathbb{N}$. Now consider the new system $\Delta' := \Delta \cup \{Z \xrightarrow{a} \epsilon\}$. The process $(X\|Z, \Delta')$ is infinite w.r.t. bisimilarity, because $(X\|Z, \Delta') \xrightarrow{a} (X, \Delta')$. We say that by this transition the subprocess X is *unmasked*. Of course, this is only a very trivial example of masking and unmasking. In general the question if a process can be unmasked (i.e., if a place matters w.r.t. bisimilarity) is $\mathcal{NP}$-hard. Later in this section we use a more complex example of masking and unmasking to prove this.

For the subclass of *normed* BPP, finiteness w.r.t. bisimilarity coincides with boundedness. Thus for *normed* BPP finiteness w.r.t. bisimilarity is decomposable into properties of subprocesses and decidable in polynomial time. In particular, for *normed* BPP, the parallel composition of two infinite processes yields an infinite process. For general BPP it is different. The parallel composition of infinite processes (w.r.t. bisimilarity) can yield a process that is finite w.r.t. bisimilarity. Thus, finiteness (or infiniteness) w.r.t. bisimilarity of a BPP process cannot be decomposed into properties of subprocesses in general. The following example shows this. Let Δ be

$$X_i \xrightarrow{a_{i+1}} X_i\|Y_i, \quad Y_i \xrightarrow{a_i} \epsilon \ \text{ for } 1 \le i \le n-1$$
$$X_n \xrightarrow{a_1} X_n\|Y_n,\ Y_n \xrightarrow{a_n} \epsilon$$

Then the process $X_1\|X_2\|\ldots\|X_n$ is finite w.r.t. bisimilarity, but every subprocess (e.g. $X_3\|X_4\|X_7$) is infinite w.r.t. bisimilarity.

Now we are ready to prove the co-$\mathcal{NP}$ lower bound for strong bisimilarity of BPP. We do this by a polynomial reduction of 3-SAT to the negation of the problem. Let $n \in \mathbb{N}$

and let $x_1, \ldots, x_n$ be boolean variables. A literal is either a variable or the negation of a variable. A clause is a disjunction of 3 literals. Let $Q := Q_1 \wedge \ldots \wedge Q_k$ be a boolean formula in 3-CNF over $x_1, \ldots, x_n$ with k clauses. We construct BPPs P_1 and P_2 s.t. Q is satisfiable iff $P_1 \not\sim P_2$. The set of transition rules Δ is defined as follows.

For every $i \in \{1, \ldots, n\}$ we have $X_i \xrightarrow{x_i} X_{i+1} \| \alpha_i$ where α_i is a parallel composition of constants defined as follows: For every $j \in \{1, \ldots, k\}$ let A_j be in α_i iff the first literal of Q_j is $\bar{x}_i$. For every $j \in \{1, \ldots, k\}$ let B_j be in α_i iff the second literal of Q_j is $\bar{x}_i$. For every $j \in \{1, \ldots, k\}$ let C_j be in α_i iff the third literal of Q_j is $\bar{x}_i$. For every $i \in \{1, \ldots, n\}$ we have $X_i \xrightarrow{\bar{x}_1} X_{i+1} \| \beta_i$ where β_i is a parallel composition of constants defined as follows: For every $j \in \{1, \ldots, k\}$ let A_j be in β_i iff the first literal of Q_j is x_i. For every $j \in \{1, \ldots, k\}$ let B_j be in β_i iff the second literal of Q_j is x_i. For every $j \in \{1, \ldots, k\}$ let C_j be in β_i iff the third literal of Q_j is x_i. The intuition is that by action $x_i/\bar{x}_i$ one chooses the value *true/false* for the variable x_i. Q is satisfiable iff the assignment of values to the variables can be chosen in such a way that for every $j \in \{1, \ldots, k\}$ at least one of the constants $\{A_j, B_j, C_j\}$ does *not* appear.

The other transition rules are as follows:

$$
\begin{array}{lll}
A_j & \xrightarrow{a_j} A_j & \text{for } 1 \leq j \leq k \\
B_j & \xrightarrow{b_j} B_j & \text{for } 1 \leq j \leq k \\
C_j & \xrightarrow{c_j} C_j & \text{for } 1 \leq j \leq k \\
X_i & \xrightarrow{d_j} X_i & \text{for } 1 \leq i \leq n+1,\ 1 \leq j \leq k,\ d_j \in \{a_j, b_j, c_j\} \\
X_{n+1} & \xrightarrow{e} Y_1 \| \ldots \| Y_k & \\
Y_j & \xrightarrow{d_j} Z_{j+1} \| W_j & \text{for } 1 \leq j \leq k,\ d_j \in \{a_j, b_j, c_j\} \\
W_j & \xrightarrow{d_l} W_j & \text{for } 1 \leq j \leq k,\ 1 \leq l \leq j,\ d_l \in \{a_l, b_l, c_l\} \\
Z_j & \xrightarrow{d_j} \epsilon & \text{for } 1 \leq j \leq k,\ d_j \in \{a_j, b_j, c_j\} \\
Z_{k+1} & \xrightarrow{\lambda} Z_{k+1} &
\end{array}
$$

Figure 2 gives a rough description of Δ in Petri net notation. Let $P_1 := (X_1 \| Z_1, \Delta)$ and $P_2 := (X_1, \Delta)$.

Lemma 2. *If Q is satisfiable then $P_1 \not\sim P_2$.*

Proof. We show that the attacker has the following winning strategy. Since Q is satisfiable, there exists an assignment of variables that makes Q true. The attacker can choose this assignment by performing the corresponding actions x_i or $\bar{x}_i$ for $1 \leq i \leq n$ in either P_1 or P_2. Then the attacker does the action e. The defender can only respond by doing exactly the same. This yields the new states P_1' and P_2' with $P_1' = P_2' \| Z_1$. For every $j \in \{1, \ldots, k\}$ there is at least one constant $D_j \in \{A_j, B_j, C_j\}$ that does not appear in P_1' or P_2'. Let d_j be the action corresponding to D_j, e.g. if $D_1 = B_1$ then $d_1 = b_1$.

The attacker performs the action d_1 by the rule $Z_1 \xrightarrow{d_1} \epsilon$ in P_1'. Since neither D_1 nor Z_1 occurs in P_2' the defender can only respond by $Y_1 \xrightarrow{d_1} Z_2 \| W_1$. Let the resulting states be P_1'' and P_2''. Now the attacker performs $Z_2 \xrightarrow{d_2} \epsilon$ in P_2'' to which the defender can only respond by $Y_2 \xrightarrow{d_2} Z_3 \| W_2$ in P_1'' and so on with Z_j, d_j for $1 \leq j \leq k$. In the end the defender is forced to perform the transition $Y_k \xrightarrow{d_k} Z_{k+1} \| W_k$. Now the action λ is

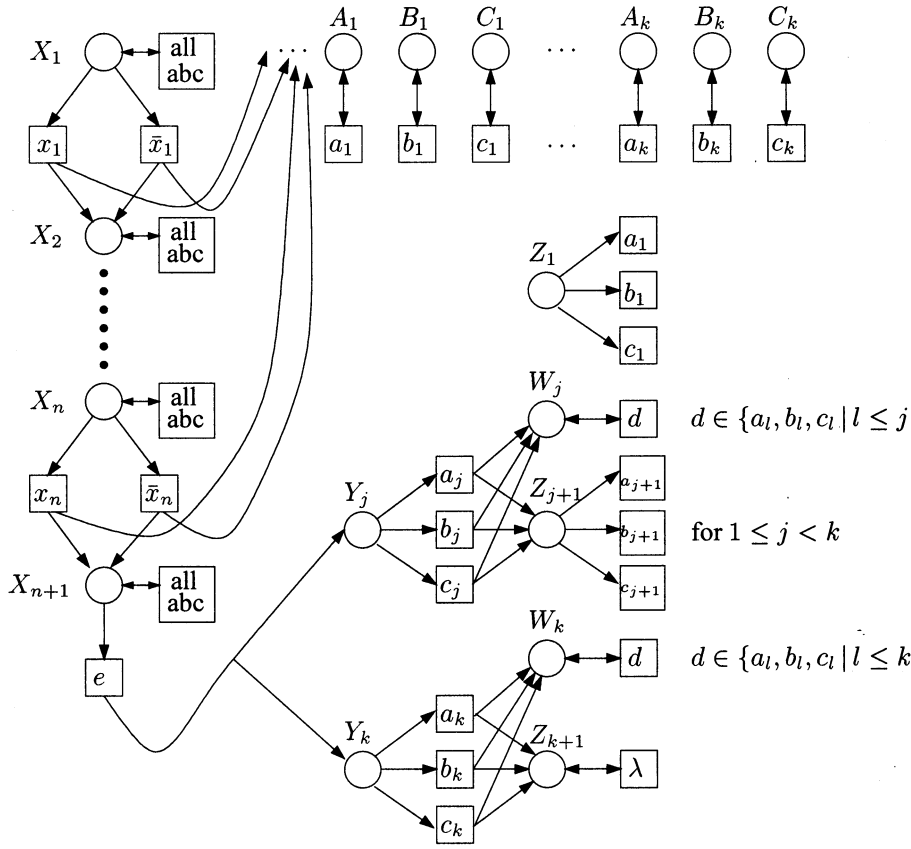


Fig. 2. Δ in Petri net notation. 'All abc' means all actions a_i, b_i, c_i for every $i \in \{1, \ldots, k\}$.

enabled in one process (by the constant Z_{k+1}), but not in the other. Thus the attacker can win and $P_1 \not\sim P_2$. □

Lemma 3. *If Q is not satisfiable then $P_1 \sim P_2$.*

Proof. Let AS (for 'assignments') be the set of subterms containing only constants A_j, B_j, C_j for $1 \leq j \leq k$. We call a term $t \in AS$ a *faulty assignment* iff there is at least one $m \in \{1, \ldots, k\}$ s.t. all three constants A_m, B_m, C_m occur in t. We call the minimal such m the *index* of t, denoted $ind(t)$. Let FAS be the set of faulty assignments. Since Q is not satisfiable, every assignment t that is created by performing one of each pair of actions $x_1/\bar{x}_1 \ldots x_n/\bar{x}_n$ is a faulty assignment. Any incomplete assignment t' that is created by an incomplete prefix of choices from $x_1/\bar{x}_1 \ldots x_j/\bar{x}_j$ (with $j < n$) must in the end become a faulty assignment once all choices from $x_1/\bar{x}_1 \ldots x_n/\bar{x}_n$ have been made. Let $IFAS_j$ be the set of these incomplete faulty assignments created by choosing one of each pair of actions $x_1/\bar{x}_1 \ldots x_j/\bar{x}_j$. Let O_j be the set of terms containing only constants Y_l, W_l, Z_l, Z_{l+1} with $l \leq j$. To keep the notation simple we define $W_0 := \epsilon$.

The symmetric closure of the following relation is a bisimulation.

$$\begin{array}{l}
\{(X_i\|t\|Z_1^u,\ X_i\|t\|Z_1^v) \mid 1 \le i \le n \wedge t \in IFAS_{i-1} \wedge u,v \in \mathbb{N}_0\} \cup \\
\{(X_{n+1}\|t\|Z_1^u,\ X_{n+1}\|t\|Z_1^v) \mid t \in FAS \wedge u,v \in \mathbb{N}_0\} \cup \\
\{(Y_1\|\ldots\|Y_k\|t\|Z_1^u,\ Y_1\|\ldots Y_k\|t\|Z_1^v) \mid t \in FAS \wedge u,v \in \mathbb{N}_0\} \cup \\
\{(Y_j\|\ldots\|Y_k\|W_{j-1}\|t\|\gamma,\ Y_{j+1}\|\ldots\|Y_k\|t\|Z_{j+1}\|W_j\|\gamma') \mid \\
t \in FAS \wedge j+1 \le ind(t) \wedge \gamma,\gamma' \in O_{j-1}\} \cup \\
\{(Y_j\|\ldots\|Y_k\|t\|W_{j-1}\|\gamma,\ Y_{j+1}\|\ldots\|Y_k\|t\|W_j\|\gamma') \mid \\
t \in FAS \wedge j+1 \le ind(t) \wedge \gamma,\gamma' \in O_{j-1}\} \cup \\
\{(Y_{j+1}\|\ldots\|Y_k\|t\|Z_{j+1}\|W_j\|\gamma,\ Y_{j+1}\|\ldots\|Y_k\|t\|W_j\|\gamma') \mid \\
t \in FAS \wedge j+1 \le ind(t) \wedge \gamma,\gamma' \in O_{j-1}\} \cup \\
\{(W_j\|Z_{j+1}^u\|Y_{j+1}\|\ldots\|Y_k\|\gamma\|t,\ W_j\|Z_{j+1}^u\|Y_{j+1}\|\ldots\|Y_k\|\gamma'\|t) \mid \\
u \in \{0,1\} \wedge t \in FAS \wedge 1 \le j \le k \wedge \gamma,\gamma' \in O_{j-1}\}
\end{array}$$

Since $(X_1\|Z_1, X_1)$ is in this relation, we get $P_1 \sim P_2$. □

Theorem 4. *Strong bisimilarity of BPP is co-$\mathcal{NP}$-hard.*
Proof. Directly from Lemma 2 and Lemma 3 and the $\mathcal{NP}$-completeness of 3-SAT.

Note that both P_1 and P_2 are bounded, i.e., they have only finitely many reachable states. It is easy to see that in general the number of reachable states of P_1/P_2 is exponential in the size of the description of Δ. Moreover, the number of reachable states of P_1/P_2 is even exponential up to strong bisimilarity, i.e., they generally have an exponential number of non-bisimilar reachable states. Let $t \in FAS$. Analogously to the definition of the index of t we define $ind'(t)$ as the *maximal* m s.t. all three A_m, B_m, C_m appear in t. Consider the reachable states $X_{n+1}\|t_1\|t_2$, where $t_1\|t_2 \in FAS$ encodes a faulty assignment and the constants A_j, B_j, C_j in t_1 have $j \le ind'(t_1\|t_2)$ and the constants A_j, B_j, C_j in t_2 have $j > ind'(t_1\|t_2)$. In particular $ind'(t_1\|t_2) = ind'(t_1)$. While the particular structure of t_1 does not matter for bisimilarity (as long as $t_1 \in FAS$), the structure of t_2 does. We have $X_{n+1}\|t_1\|t_2 \not\sim X_{n+1}\|t_1\|t_2'$ for every $t_2' \neq t_2$. Since there are in general exponentially many different such t_2' it follows that P_1/P_2 is at least exponential w.r.t. strong bisimilarity. Thus, our construction does not yield a lower bound for the problem of strong bisimilarity of a BPP and a finite-state process (with polynomially many states). It seems to be impossible to prove a lower bound for this asymmetric problem, since whenever one encodes a sufficiently complex problem (e.g. SAT) into a BPP, this BPP is never bisimilar (neither strongly nor weakly) to any finite-state system of polynomial size (although it can be bisimilar to a finite-state system of exponential size). Thus, we conjecture that strong and weak bisimilarity of a BPP and a finite-state system is decidable in polynomial time. (Is is known that strong and weak bisimilarity of a *normed* BPP and a finite-state system is polynomial [18]).

Now we consider the strong finiteness problem.

STRONG FINITENESS OF BPP

Instance: A BPP process P.
Question: Does there exist a finite-state system F s.t. $P \sim F$?

Finiteness w.r.t. strong bisimilarity is decidable even for general Petri nets [14], and this result carries over immediately to communication-free nets (i.e., BPP). However, the

algorithm in [14] consists of two semidecision procedures and gives no upper bound on the complexity. For general Petri nets one gets an $EXPSPACE$ lower bound by reducing the problem if a given place can ever become marked to the finiteness problem. For general Petri nets the problem if a given place can ever become marked is $EXPSPACE$-hard [19, 28]. For communication-free nets (i.e., BPP) this is different. While the reachability problem is $\mathcal{NP}$-complete for communication-free nets [8], it is easy to see that the problem if a given place can ever become marked in a communication-free net is polynomial. Thus, one does not obtain a lower bound for the strong finiteness problem of BPP that way.

It is clear that a *constructive* solution to the problem, i.e., constructing the finite-state system F if it exists, must require at least exponential time. This is because there are BPPs s.t. the smallest finite-state system F that is bisimilar to them has an exponential number of states (in the size of the description of the BPP). However, it is not immediately clear if a simple yes/no answer to the strong finiteness problem must be as hard. The following theorem shows this.

Theorem 5. *Strong finiteness of BPP is co-$\mathcal{NP}$-hard.*

Proof. By a polynomial reduction of 3-SAT to strong infiniteness. Let the formula Q and the set of rules Δ be defined as before and let $\Delta' := \Delta \cup \{X_{n+1} \xrightarrow{p} X_{n+1} \| Z_1\}$. We show that the process X_1 w.r.t. the set of rules Δ', denoted (X_1, Δ'), is infinite w.r.t. strong bisimilarity iff Q is satisfiable.

$\Leftarrow$ If Q is satisfiable then there are infinitely many reachable states $Y_1 \| \ldots \| Y_k \| \gamma \| Z_1^m$ for every $m \in \mathbb{N}_0$, where γ is a term that encodes a satisfying assignment of Q. This means that γ is a parallel composition of constants A_j, B_j, C_j where for every $j \in \{1, \ldots, k\}$ at least one of the constants A_j, B_j, C_j does not occur in γ. However, for every $m_1 \neq m_2$ we have $Y_1 \| \ldots \| Y_k \| \gamma \| Z_1^{m_1} \not\sim Y_1 \| \ldots \| Y_k \| \gamma \| Z_1^{m_2}$, because the attacker has a winning strategy similar to the one in Lemma 2. Thus (X_1, Δ') is infinite w.r.t. strong bisimilarity.

$\Rightarrow$ Let $\Delta'' := \Delta \cup \{X_{n+1} \xrightarrow{p} X_{n+1}\}$. The process (X_1, Δ'') has finitely many reachable states. (However, (X_1, Δ'') has an exponential (in the size of Δ'') number of non-bisimilar reachable states.) If Q is not satisfiable then $(X_1, \Delta') \sim (X_1, \Delta'')$ and is thus finite w.r.t. bisimilarity. The bisimulation relation is the same as in Lemma 3.

□

The previous construction shows that the problem if a place can be unmasked (i.e., made to count w.r.t. bisimulation) is $\mathcal{NP}$-hard. Here this particular place was Z_1.

4 Hardness of Weak Bisimilarity for BPP

WEAK BISIMILARITY OF BPP

Instance: Two BPP processes P_1 and P_2.
Question: $P_1 \approx P_2$?

It is still an open question if this problem is decidable. It has been shown to be semidecidable in [8], using the facts that weak bisimulation equivalence on BPPs is

semilinear (since it is a congruence on a finitely generated commutative semigroup) and that it is decidable if a given semilinear relation on a BPP is a weak bisimulation. An $\mathcal{NP}$ lower bound for this problem has been shown in [31] (by reduction of a variant of the bin-packing problem), and the co-$\mathcal{NP}$ lower bound of Theorem 4 carries over immediately to weak bisimilarity. Here we prove a Π_2^p-lower bound (in the polynomial hierarchy) that subsumes these results.

Let $Q := Q_1 \wedge \ldots \wedge Q_k$ be a boolean formula in 3-CNF over the boolean variables $x_1, \ldots, x_n, y_1, \ldots, y_n$ with k clauses. We construct BPP processes P_1, P_2 s.t. $P_1 \approx P_2$ iff $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$. Since this problem is Π_2^p-complete, we get a Π_2^p-lower bound for the problem of weak bisimilarity.

Let α_i be a parallel composition of constants in $\{Q_1, \ldots, Q_k\}$ s.t. constant Q_j appears in α_i iff x_i makes clause Q_j true (i.e., x_i appears positively in Q_j). Let β_i be a parallel composition of constants in $\{Q_1, \ldots, Q_k\}$ s.t. constant Q_j appears in β_i iff $\bar{x}_i$ makes clause Q_j true (i.e., x_i appears negatively in Q_j). Let γ_i be a parallel composition of constants in $\{Q_1, \ldots, Q_k\}$ s.t. constant Q_j appears in γ_i iff y_i makes clause Q_j true. Let δ_i be a parallel composition of constants in $\{Q_1, \ldots, Q_k\}$ s.t. constant Q_j appears in δ_i iff $\bar{y}_i$ makes clause Q_j true. The set of transition rules Δ is defined by

$$
\begin{array}{lll}
X_i & \xrightarrow{x_i} X_{i+1} \| \alpha_i & \text{for } 1 \leq i \leq n \\
X_i & \xrightarrow{\bar{x}_i} X_{i+1} \| \beta_i & \text{for } 1 \leq i \leq n \\
X'_i & \xrightarrow{x_i} X'_{i+1} \| \alpha_i & \text{for } 1 \leq i \leq n \\
X'_i & \xrightarrow{\bar{x}_i} X'_{i+1} \| \beta_i & \text{for } 1 \leq i \leq n \\
X_{n+1} & \xrightarrow{a} Y_1 \| \ldots \| Y_n & \\
X'_{n+1} & \xrightarrow{a} Y_1 \| \ldots \| Y_n & \\
X'_{n+1} & \xrightarrow{a} Z & \\
Y_i & \xrightarrow{\tau} \gamma_i & \text{for } 1 \leq i \leq n \\
Y_i & \xrightarrow{\tau} \delta_i & \text{for } 1 \leq i \leq n \\
Q_j & \xrightarrow{q_j} Q_j & \text{for } 1 \leq j \leq k \\
Z & \xrightarrow{q_j} Z & \text{for } 1 \leq j \leq k
\end{array}
$$

Let $P_1 := (X_1, \Delta)$ and $P_2 := (X'_1, \Delta)$.

Lemma 6. *If $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$ is false then $P_1 \not\approx P_2$.*
Proof. If $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$ is false then $\exists(x_1, \ldots, x_n)\forall(y_1, \ldots, y_n)\neg Q$. The attacker chooses these values for $x_1, \ldots, x_n$ by choosing $x_i/\bar{x}_i$. The defender can only copy these moves. Then the attacker chooses the transition $X'_{n+1} \xrightarrow{a} Z$. The defender can only respond by $X_{n+1} \xrightarrow{a} Y_1 \| \ldots \| Y_n$ and then a sequence of silent τ-actions ending in a state t. By definition of Δ and since $\exists(x_1, \ldots, x_n)\forall(y_1, \ldots, y_n)\neg Q$ there will be at least one action q_j (with $1 \leq j \leq k$) that is not enabled by t (and cannot made to be enabled by τ-moves). However, all q_j are enabled by Z. Thus, the attacker has a winning strategy and $P_1 \not\approx P_2$. □

Lemma 7. *If $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$ then $P_1 \approx P_2$.*
Proof. The attacker can choose the assignment for $x_1, \ldots, x_n$. The defender can only imitate these choices. If the attacker chooses the transition $X_{n+1} \xrightarrow{a} Y_1 \| \ldots \| Y_n$ or

$X'_{n+1} \xrightarrow{a} Y_1 \| \ldots \| Y_n$ then the defender can respond in such a way that the two processes become equal and the defender wins. If the attacker chooses $X'_{n+1} \xrightarrow{a} Z$ then the defender can (by a long internal move of τ-actions) choose the values for $y_1, \ldots, y_n$ on his side. Since $\forall(x_1, \ldots, x_n)\exists(y_1, \ldots, y_n)\, Q$ there are choices for $y_1, \ldots, y_n$ s.t. in the resulting state all actions $q_1, \ldots, q_k$ are permanently enabled. Since $q_1, \ldots, q_k$ are also permanently enabled by Z in the other process and all other actions are not, the defender wins. Thus, the defender has a winning strategy and $P_1 \approx P_2$. □

Theorem 8. *Weak bisimilarity of BPP is Π_2^p-hard.*
Proof. Directly from Lemma 6 and Lemma 7.

WEAK FINITENESS OF BPP

Instance: A BPP process P.
Question: Does there exist a finite-state system F s.t. $P \approx F$?

We show that the weak finiteness problem for BPP is also Π_2^p-hard by using the previously defined processes P_1 and P_2 and constructing a new process P that is weakly finite iff $P_1 \approx P_2$. Let Δ' be $\Delta \cup \Gamma$, where Γ is the following set of transition rules:

$$I \xrightarrow{\tau} I\|C \qquad I \xrightarrow{\tau} \epsilon \qquad C \xrightarrow{c} \epsilon \qquad D \xrightarrow{c} E \qquad D \xrightarrow{c} E' \qquad D \xrightarrow{e} X_1\|S$$
$$D \xrightarrow{e} X'_1\|S \qquad E \xrightarrow{c} E \qquad E' \xrightarrow{c} E' \qquad E \xrightarrow{e} X_1\|S \qquad E' \xrightarrow{e} X'_1\|S \qquad S \xrightarrow{c} S$$

Let $P := (I\|D, \Delta')$.

Lemma 9. *If $P_1 \not\approx P_2$ then P is not weakly finite.*
Proof. P has infinitely many non-weakly-bisimilar states $D\|C^i$ for all $i \in \mathbb{N}$. It suffices to show that $D\|C^j \not\approx D\|C^i$ for $j > i$. The attacker has the following winning strategy. He does action c exactly $i+1$ times in $D\|C^j$ and reaches the state $D\|C^{j-i-1}$. The defender can respond in different ways in $D\|C^i$, but the reached state will always be either $E\|C^k$ or $E'\|C^k$ for some $k \leq i$. In the first case the attacker does the transition $D \xrightarrow{e} X'_1\|S$. The defender can only respond by $E \xrightarrow{e} X_1\|S$ and the new state in the bisimulation game is $(X'_1\|S\|C^{j-i-1}, X_1\|S\|C^k)$. This is not weakly bisimilar, because $P_1 \not\approx P_2$. The second case is symmetric with X_1 and X'_1 exchanged. □

Lemma 10. *If $P_1 \approx P_2$ then P is weakly finite.*
Proof. Let Γ' be Γ where X'_1 is replaced by X_1 and $\Delta'' := \Delta \cup \Gamma'$. Since $P_1 \approx P_2$ and weak bisimilarity is a congruence on BPP, we get $P = (I\|D, \Delta') \approx (I\|D, \Delta'')$. It is easy to see that $(I\|D, \Delta'') \approx (E, \Delta'')$, because $S \xrightarrow{c} S$. Thus $P \approx (E, \Delta'')$. However, (E, Δ'') has only finitely many reachable states. □

Theorem 11. *Weak finiteness of BPP is Π_2^p-hard.*
Proof. By Lemmas 6, 7, 9 and 10. □

5 Conclusion

The following table summarizes known results about the complexity of bisimulation problems for several classes of infinite-state systems. New results are in boldface. The different columns in the table below show the results about the following problems: strong bisimilarity with finite automata, strong bisimilarity of two infinite-state systems, weak bisimilarity with finite automata and weak bisimilarity of two infinite-state systems.

	$\sim F$	$\sim$	$\approx F$	$\approx$
FS	$\mathcal{P}$ [2, 27]	$\mathcal{P}$ [2, 27]	$\mathcal{P}$ [2, 27]	$\mathcal{P}$ [2, 27]
BPA	$\mathcal{P}$ [18]	$\in 2-EXPTIME$ [3]	$\mathcal{P}$ [18]	$PSPACE$-hard [31]
PDA	$\in EXPTIME$ [15] $PSPACE$-hard [23]	decidable [29] $PSPACE$-hard [23]	$\in EXPTIME$ [15] $PSPACE$-hard [23]	$PSPACE$-hard [31]
BPP	$\in PSPACE$ [15]	decidable [7] **co-NP-hard**	$\in PSPACE$ [15]	$\mathbf{\Pi_2^p}$**-hard**
PA	decidable [15]	**co-NP-hard**	decidable [15]	$PSPACE$-hard [31]
PAD	decidable [15] $PSPACE$-hard [23]	$PSPACE$-hard [23]	decidable [15] $PSPACE$-hard [23]	$PSPACE$-hard [31]
PN	decidable [16, 15] $EXPSPACE$-hard	undecidable [13]	undecidable [13]	undecidable [13]
PAN	$EXPSPACE$-hard	undecidable [13]	undecidable [13]	undecidable [13]
PRS	$EXPSPACE$-hard	undecidable [13]	undecidable [13]	undecidable [13]

The following table summarizes results about the problems of strong and weak finiteness. New results are in boldface.

	strong finiteness	weak finiteness
BPA	$\in 2-EXPTIME$ [4, 3]	?
PDA	$PSPACE$-hard [23]	$PSPACE$-hard [23]
BPP	decidable [14] **co-NP-hard**	$\mathbf{\Pi_2^p}$**-hard**
PA	**co-NP-hard**	$\mathbf{\Pi_2^p}$**-hard**
PAD	$PSPACE$-hard [23]	$PSPACE$-hard [23]
PN	decidable [14] $EXPSPACE$-hard	undecidable [14]
PAN/PRS	$EXPSPACE$-hard	undecidable [14]

Some more results are known about the restricted subclasses of these systems that satisfy the 'normedness condition' (e.g. [12, 11, 10, 17, 18]).

References

[1] J.C.M. Baeten and W.P. Weijland. Process algebra. *Cambridge Tracts in Theoretical Computer Science*, 18, 1990.

[2] J. Balcazar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4:638–648, 1992.

[3] O. Burkart, D. Caucal, and B. Steffen. An elementary bisimulation decision procedure for arbitrary context-free processes. In *MFCS'95*, volume 969 of *LNCS*. Springer Verlag, 1995.

[4] O. Burkart, D. Caucal, and B. Steffen. Bisimulation collapse and the process taxonomy. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.
[5] D. Caucal. On the regular structure of prefix rewriting. *Journal of Theoretical Computer Science*, 106:61–86, 1992.
[6] S. Christensen. *Decidability and Decomposition in Process Algebras*. PhD thesis, Edinburgh University, 1993.
[7] S. Christensen, Y. Hirshfeld, and F. Moller. Bisimulation equivalence is decidable for Basic Parallel Processes. In E. Best, editor, *Proceedings of CONCUR 93*, volume 715 of *LNCS*. Springer Verlag, 1993.
[8] J. Esparza. Petri nets, commutative context-free grammars and Basic Parallel Processes. In Horst Reichel, editor, *Fundamentals of Computation Theory*, volume 965 of *LNCS*. Springer Verlag, 1995.
[9] J. Esparza and A. Kiehn. On the model checking problem for branching time logics and Basic Parallel Processes. In *CAV'95*, volume 939 of *LNCS*, pages 353–366. Springer Verlag, 1995.
[10] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. of ICALP'99*, volume 1644 of *LNCS*. Springer Verlag, 1999.
[11] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity of normed context-free processes. *Theoretical Computer Science*, 158:143–159, 1996.
[12] Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial-time algorithm for deciding bisimulation equivalence of normed Basic Parallel Processes. *Journal of Mathematical Structures in Computer Science*, 6:251–259, 1996.
[13] P. Jančar. Undecidability of bisimilarity for Petri nets and some related problems. *Theoretical Computer Science*, 148:281–301, 1995.
[14] P. Jančar and J. Esparza. Deciding finiteness of Petri nets up to bisimulation. In F. Meyer auf der Heide and B. Monien, editors, *Proceedings of ICALP'96*, volume 1099 of *LNCS*. Springer Verlag, 1996.
[15] P. Jančar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state processes. In *Proc. of ICALP'98*, volume 1443 of *LNCS*. Springer Verlag, 1998.
[16] P. Jančar and F. Moller. Checking regular properties of Petri nets. In Insup Lee and Scott A. Smolka, editors, *Proceedings of CONCUR'95*, volume 962 of *LNCS*. Springer Verlag, 1995.
[17] A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.
[18] A. Kučera and R. Mayr. Weak bisimilarity with infinite-state systems can be decided in polynomial time. In *Proc. of CONCUR'99*, volume 1664 of *LNCS*. Springer Verlag, 1999.
[19] R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.
[20] R. Mayr. Weak bisimulation and model checking for Basic Parallel Processes. In *Foundations of Software Technology and Theoretical Computer Science (FST&TCS'96)*, volume 1180 of *LNCS*. Springer Verlag, 1996.
[21] R. Mayr. Tableau methods for PA-processes. In D. Galmiche, editor, *Analytic Tableaux and Related Methods (TABLEAUX'97)*, volume 1227 of *LNAI*. Springer Verlag, 1997.
[22] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU-München, 1998.
[23] R. Mayr. On the complexity of bisimulation problems for pushdown automata. 2000.
[24] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286, 2000.
[25] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
[26] F. Moller. Infinite results. In Ugo Montanari and Vladimiro Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *LNCS*. Springer Verlag, 1996.

[27] R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.
[28] J.L. Peterson. *Petri net theory and the modeling of systems*. Prentice-Hall, 1981.
[29] G. Sénizergues. Decidability of bisimulation equivalence for equational graphs of finite out-degree. In *Proc. of FOCS'98*. IEEE, 1998.
[30] C. Stirling. The joys of bisimulation. In *Proc. of MFCS'98*, volume 1450 of *LNCS*, pages 142–151. Springer Verlag, 1998.
[31] J. Stříbrná. Hardness results for weak bisimilarity of simple process algebras. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 18, 1998.
[32] W. Thomas. On the Ehrenfeucht-Fraïssé game in theoretical computer science. In *Proc. of TAPSOFT'93*, volume 668 of *LNCS*, pages 559–568. Springer Verlag, 1993.

Decidable First-Order Transition Logics for PA-Processes

Denis Lugiez[1] and Philippe Schnoebelen[2]

[1] Lab. d'Informatique de Marseille, Univ. Aix-Marseille 1 & CNRS URA 1787,
39, r. Joliot-Curie, 13453 Marseille Cedex 13, France
lugiez@lim.univ-mrs.fr
[2] Lab. Spécification et Vérification, ENS de Cachan & CNRS UMR 8643,
61, av. Pdt Wilson, 94235 Cachan, France
phs@lsv.ens-cachan.fr

Abstract. We show decidability of several first-order logics based upon the reachability predicate in PA. The main tool we use is the recognizability by tree automata of the reachability relation between PA-processes. This approach and the transition logics we use allow a smooth and general treatment of parameterized model checking for PA. Then the logic is extended to handle a general notion of costs of PA-steps. In particular, when costs are Parikh images of traces, we show decidability of a transition logic extended by some form of first-order reasoning over costs.

1 Introduction

PA [BW90] is the formal model of concurrency allowing recursive definitions, sequential and parallel compositions, but where actions are uninterpreted and do not synchronize. PA gives rise to infinite-state systems and is quite expressive. Recently, several verification problems have been shown decidable for PA using a variety of fairly involved techniques [1]. Decidability results for PA are mainly theoretical, and they help delineate some important frontiers in the field of verification for infinite-state processes. In [LS99], we advocated *regular tree languages* and *tree automata* as an easy-to-use tool for tackling problems about PA, and we proved that the reachability sets of a regular set of PA processes is regular. Our proofs are effective and give simple polynomial-time algorithms which can be used for a variety of problems based on reachability among PA-processes. [EK99,EP00] showed how to use and adapt them for problems in static analysis.

Contents of the paper. Here we extend our previous work in several ways:

Recognizable tree relations: We replace automata for tree languages by automata for tree relations and show that $\xrightarrow{*}$ over PA-processes is recognizable.

First-order transition logic: This gives a decision method for the first-order transition logic, i.e. the first-order logic having $\rightarrow$, $\xrightarrow{*}$, and equality as basic

[1] See, e.g., [BEH95b,BEH95a,Kuč96,Kuč97,JKM98,HJ99,May99].

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 342–353, 2000.
© Springer-Verlag Berlin Heidelberg 2000

predicates (plus any other recognizable predicates). The method computes the set of solutions of a given formula, and thus allows parameterized model checking, model measuring, ...

Costs: We enrich PA with a notion of "cost of steps" which is more general than traces. These costs can encode various measures and view PA as a truly concurrent model (e.g., costs can encode timing measures where parallelism is faster than interleaving). We extend the transition logic with decomposable cost predicates and show the decidability of several timed transition logics.

Parameterized constraints over $\xrightarrow{*}$: Finally, we define TLC, the transition logic where costs are the Parikh images of traces and where integer variables and Presburger formulas are freely used to state constraints on reachability. TLC is not decidable but we isolate a rich fragment which is.

Related work. Several temporal logics with cost constraints have been proposed for finite state systems (see [AELP99,ET99] for recent proposals). Some fragments of the temporal logics from [BEH95b,BEH95a] apply to PA but temporal logics deal with paths and are quite different from transition logics where a first-order theory of *states* is available (more explanations in section 7.1).

For costs that are Parikh images of traces, [Esp97] shows recognizability of the ternary relation $s \xrightarrow{c} t$ over BPP (PA without sequential composition) but does not consider applications to the first-order transition logic. [CJ99] shows recognizability of the reachability relation between configurations of timed automata, introduces the transition logic and uses it for model measuring and parameterized model checking. An important technical difference is that our automata recognize pairs of trees (PA-processes) while [Esp97] handles tuples of integers (markings of BPP's) and [CJ99] handles tuples of reals (clock values).

Reachability in PA is investigated in [May97,May99]. The underlying methods apply to more general systems (like PRS [Mol96]) but they are quite complex since they view terms modulo structural congruence. As explained in [LS99], we believe it is better to only introduce structural congruence at a later stage.

The combination of costs and tree automata is studied by Seidl [Sei94] for compiler optimization (using more general costs than ours), not decidability of logics on trees (where the relevant problem is the combination of cost automata).

2 The PA Process Algebra

PA may be defined in several (equivalent) ways. Our definition:

(1) uses rewrite rules *à la* Moller [Par66],
(2) does not identify terms modulo structural congruence,
(3) incorporates a notion of costs for steps,
(4) is a *big-steps* semantics (in the sense of [Plo81]).

Syntax. We assume $Act = \{a, b, \dots\}$ is a finite set of *action names* and write $Act^* = \{w, \dots\}$ for the set of words over Act (with empty string denoted ε).

For $w_1, w_2 \in Act^*$, we let $w_1 \parallel w_2$ denote their shuffle product, i.e. the set of all their interleavings.

We assume $M = \{c, \dots\}$ is a set of values, called *costs*, equipped with two binary associative-commutative operations $\oplus$ and $\otimes$ with same neutral element 0_M (we see several different examples of cost sets in sections 6, 7). Given a set $Const = \{X, Y, Z, \dots\}$ of *process constants* (or names), $\mathcal{T}_{Const}$, or $\mathcal{T}$ when the underlying $Const$ is clear, is the set $\{s, t, \dots\}$ of *PA-terms*, given by the following abstract syntax

$$s, t ::= s.t \;\mid\; s \parallel t \;\mid\; O \;\mid\; X \;\mid\; Y \;\mid\; Z \;\mid\; \cdots$$

For $t \in \mathcal{T}$, $Const(t)$ denotes the set of process constants occurring in t. A PA *declaration* is a finite $Const$ with a finite set $\Delta \subseteq Const \times Act \times M \times \mathcal{T}$ of *process rewrite rules*. A rule $(X, a, c, t) \in \Delta$ is written $X \xrightarrow{a,c} t$. For simplicity, we require that all $X \in Const$ appear in the left-hand side of at least one rule.

For $t \in \mathcal{T}$, we let $Sub(t)$ denote the set of all subterms of t. Similarly, we write $Sub(\Delta)$ for the finite set of all subterms of (some term from) Δ. The size of a term is $|t| \stackrel{\text{def}}{=} Card(Sub(t))$ and the size of a PA declaration is $|\Delta| \stackrel{\text{def}}{=} Card(Sub(\Delta))$.

Semantics. A PA declaration Δ defines a labeled transition system $(\mathcal{T}, \rightarrow)$ with $\rightarrow \subseteq \mathcal{T} \times Act^* \times M \times \mathcal{T}$. We write $s \xrightarrow{w,c} t$ when $(s, w, c, t) \in \rightarrow$. The transition relation $\xrightarrow{w,c}$ is defined by the following SOS rules:

$$(\mathrm{R}_\varepsilon) \quad \frac{}{0 \xrightarrow{\varepsilon, 0_M} 0} \qquad (\mathrm{R}_\mathrm{C}) \quad \frac{t \xrightarrow{w,c'} t'}{X \xrightarrow{aw, c \oplus c'} t'} \text{ if } X \xrightarrow{a,c} t \in \Delta$$

$$(\mathrm{R}'_\varepsilon) \quad \frac{}{X \xrightarrow{\varepsilon, 0_M} X} \qquad (\mathrm{R}_\mathrm{P}) \quad \frac{t_1 \xrightarrow{w_1,c_1} t'_1 \quad t_2 \xrightarrow{w_2,c_2} t'_2}{t_1 \parallel t_2 \xrightarrow{w, c_1 \otimes c_2} t'_1 \parallel t'_2} \text{ if } w \in w_1 \parallel w_2$$

$$(\mathrm{R}_\mathrm{S}) \quad \frac{t_1 \xrightarrow{w_1,c_1} t'_1}{t_1.t_2 \xrightarrow{w_1,c_1} t'_1.t_2} \qquad (\mathrm{R}'_\mathrm{S}) \quad \frac{t_1 \xrightarrow{w_1,c_1} t'_1 \quad t_2 \xrightarrow{w_2,c_2} t'_2}{t_1.t_2 \xrightarrow{w_1 w_2, c_1 \oplus c_2} t'_1.t'_2} \text{ if } Const(t'_1) = \varnothing$$

The intuition formalized by $s \xrightarrow{w,c} t$ is that s can evolve into t by performing the sequence of actions w and the cost of that derivation is c. In general there may exist several different derivations between a s and a t: they may have same cost or not, and use same sequence of actions or not. For instance, if $\Delta = \{X \xrightarrow{a,c} Y, X \xrightarrow{b,c'} Y\}$, then the cost of reaching Y from X is either c or c'.

We write $s \xrightarrow{w} t$ (or $s \xrightarrow{c} t$, or $s \xrightarrow{*} t$) when $s \xrightarrow{w,c} t$ for some c (resp. for some w, for some w, c). Even though we started with a big-steps semantics, it is convenient to have small-steps too and we write $s \rightarrow t$ when $s \xrightarrow{w} t$ for some w of length 1. We also use $\xrightarrow{+}$ defined as $\rightarrow \circ \xrightarrow{*}$.

For $t \in \mathcal{T}$, the set $Post^*(t) \stackrel{\text{def}}{=} \{t' \mid t \xrightarrow{*} t'\}$ and $Pre^*(t) \stackrel{\text{def}}{=} \{t' \mid t' \xrightarrow{*} t\}$ denote the set of *iterated successors* and (resp.) *iterated predecessors* of t. $Post^*(t)$ is also called the reachability set of t.

3 Tree Automata and Regular Cost Grammars

Given a finite ranked alphabet $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_m$, $T_{\mathcal{F}}$ denotes the set of terms (or finite trees) built from $\mathcal{F}$. A *tree language* is any subset L of $T_{\mathcal{F}}$.

3.1 Tree Automata and Regular Tree Grammars

Tree automata recognize sets of trees (see [CDG+99]). Formally, a *tree automaton* is a tuple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \delta \rangle$ where $\mathcal{F}$ is a finite ranked alphabet, $\mathcal{Q} = \{q_1, \ldots, \}$ is a finite set of states, and $\delta \subseteq \cup_{n \in \mathbb{N}} (\mathcal{F}_n \times \mathcal{Q}^n \times \mathcal{Q})$ is a finite set of *transition rules*. A rule $\langle f, q_1, \ldots, q_n, q \rangle \in \delta$ is usually written $f(q_1, \ldots, q_n) \longmapsto q$ and is read as "*if the subterms $t_1, \ldots, t_n$ of some $t = f(t_1, \ldots, t_n)$ have been labeled by $q_1, \ldots, q_n$, then $\mathcal{A}$ may label t with q*". Given a term t, the automaton labels the nodes of t by states according to the rules in a bottom-up way. We write $t \overset{*}{\longmapsto} q$ when $t \in T_{\mathcal{F}}$ may be rewritten into $q \in \mathcal{Q}$ using the rules of δ.

Recognizability. When $t \overset{*}{\longmapsto} q$, we say t is *recognized* (also *accepted*) by state q. We write $L(q)$ for $\{t \mid t \overset{*}{\longmapsto} q\}$ and say it is the tree language recognized by q. If we add a set $F \subseteq \mathcal{Q}$ of *final states* to some tree automaton $\mathcal{A}$, then the language recognized by $\mathcal{A}$ is $L(\mathcal{A}) \overset{\text{def}}{=} \cup_{q \in F} L(q) = \{t \mid \exists q \in F, t \overset{*}{\longmapsto} q\}$. We say that $L \subseteq T_{\mathcal{F}}$ is *recognizable* if $L = L(\mathcal{A})$ for some tree automaton. Recognizable tree languages are closed under union, intersection, complementation.

ε-rules. Tree automata with ε-rules further allow rules of the form $q \longmapsto q'$. These rules may be used anywhere inside $t \overset{*}{\longmapsto} q''$. With the classical subset construction, tree automata with ε-rules can be transformed into equivalent deterministic tree automata without ε-rules.

Top-down tree automata are structurally bottom-up tree automata but label trees in a top-down way. The difference is that a rule is now written $q \to f(q_1, \ldots, q_n)$ (or $q' \to q$ for an ε-rule) and that the final states are now called *initial states*. The most important difference between top-down and bottom-up automata is a difference in viewpoint. Top-down tree automata are often seen as regular tree grammars, i.e. as generators rather than acceptors.

3.2 Recognizability of $Post^*(X)$

We now show that, for any PA-declaration Δ, and any $X \in Const$, the set $Post^*(X)$ is a regular tree language and give at the same time a grammar for the costs of the derivations. This generalizes the result of [LS99] to costs and uses a slightly different approach which yields the proof that the relation $\overset{*}{\to}$ is recognizable.

Given a cost set $\langle M, \oplus, \otimes \rangle$, a *regular cost grammar* over M is a set $\mathcal{C}$ of non-terminals $C_1, \ldots, C_n$ together with cost rules $C_{n_i} \to E_{n_i}$ where the E's are simple right-hand sides of the form "$C \oplus C'$" or "$C \otimes C'$" (where C, C' are non-terminals), or "c" (where $c \in M$). Cost grammars describe subsets of M: the rules define a set of recursive inclusions that admit a least fixpoint solution. For simplicity we write C for the subset of M defined by the non-terminal C and use shortcuts like $C \to C'$ (ε-rules) or $C \to c \otimes C'$ for grammar rules.

A regular tree grammar and a regular cost grammar for $Post^*(X)$

$$\left.\begin{array}{rcl} I_0 & \to & 0 \\ Q_0 & \to & 0 \\ Q'_0 & \to & 0 \end{array}\quad\begin{array}{rcl} C_0^I & \to & 0_M \\ C_0^Q & \to & 0_M \\ C_0^{Q'} & \to & 0_M \end{array}\right\}\text{ if } 0 \in Sub(\Delta)$$

$$\left.\begin{array}{rcl} I_Y & \to & Y \\ Q_Y & \to & Y \end{array}\quad\begin{array}{rcl} C_Y^I & \to & 0_M \\ C_Y^Q & \to & 0_M \end{array}\right\}\text{ for all } Y \in Sub(\Delta)$$

$$\left.\begin{array}{rcl} Q_Y & \to & Q_s \\ Q'_Y & \to & Q'_s \end{array}\quad\begin{array}{rcl} C_Y^Q & \to & c \oplus C_s^Q \\ C_Y^{Q'} & \to & c \oplus C_s^{Q'} \end{array}\right\}\text{ for all } Y \xrightarrow{a,c} s \in \Delta$$

$$\left.\begin{array}{rcl} I_{s_1 \| s_2} & \to & I_{s_1} \| I_{s_2} \\ Q_{s_1 \| s_2} & \to & Q_{s_1} \| Q_{s_2} \\ Q'_{s_1 \| s_2} & \to & Q'_{s_1} \| Q'_{s_2} \end{array}\quad\begin{array}{rcl} C_{s_1 \| s_2}^I & \to & C_{s_1}^I \otimes C_{s_2}^I \\ C_{s_1 \| s_2}^Q & \to & C_{s_1}^Q \otimes C_{s_2}^Q \\ C_{s_1 \| s_2}^{Q'} & \to & C_{s_1}^{Q'} \otimes C_{s_2}^{Q'} \end{array}\right\}\text{ for all } s_1 \| s_2 \in Sub(\Delta)$$

$$\left.\begin{array}{rcl} I_{s_1.s_2} & \to & I_{s_1}.I_{s_2} \\ Q_{s_1.s_2} & \to & Q_{s_1}.I_{s_2} \\ & \mid & Q'_{s_1}.Q_{s_2} \\ Q'_{s_1.s_2} & \to & Q'_{s_1}.Q'_{s_2} \end{array}\quad\begin{array}{rcl} C_{s_1.s_2}^I & \to & C_{s_1}^I \oplus C_{s_2}^I \\ C_{s_1.s_2}^Q & \to & C_{s_1}^Q \oplus C_{s_2}^I \\ & \mid & C_{s_1}^{Q'} \oplus C_{s_2}^Q \\ C_{s_1.s_2}^{Q'} & \to & C_{s_1}^{Q'} \oplus C_{s_2}^{Q'} \end{array}\right\}\text{ for all } s_1.s_2 \in Sub(\Delta)$$

These rules denote both a tree automaton $\mathcal{A}_{Post^*}$, and a cost grammar $\mathcal{C}_{Post^*}$. Their relationship is stated in the following proposition:

Proposition 3.1. *For all $t \in Sub(\Delta)$:*

1. *$I_t \xrightarrow{*} s$ iff $s = t$ and $t \xrightarrow{0_M} s$. Furthermore, $C_t^I = \{0_M\}$.*
2. *$Q_t \xrightarrow{*} s$ iff $t \xrightarrow{*} s$. Furthermore, if $t \xrightarrow{c} s$ then $c \in C_t^Q$, and if $c \in C_t^Q$ then $t \xrightarrow{c} s'$ for some s' such that $Q_t \xrightarrow{*} s'$.*
3. *$Q'_t \xrightarrow{*} s$ iff $t \xrightarrow{*} s$ and s is terminated. Furthermore, if $t \xrightarrow{c} s$ then $c \in C_t^{Q'}$, and if $c \in C_t^{Q'}$ then $t \xrightarrow{c} s'$ for some terminated s' such that $Q'_t \xrightarrow{*} s'$.*

4 Tree Automata and n-ary Relations

Products of trees. We follow [DT90]. Given two terms $s, t \in T_{\mathcal{F}}$, the pair (s, t) can be seen as one term over a product alphabet $\mathcal{F}_\times \stackrel{\text{def}}{=} (\mathcal{F} \cup \{\bot\}) \times (\mathcal{F} \cup \{\bot\}) - \{\bot\bot\}$ where $\bot$ is a new symbol with arity 0. In $\mathcal{F}_\times$ the arity of fg is the maximum of the arities of f and g. Formally we define $s \times t$ as the term in $T_{\mathcal{F}_\times}$ given recursively by

$$f(s_1, \ldots, s_n) \times g(t_1, \ldots, t_m) \stackrel{\text{def}}{=} \begin{cases} fg(s_1 \times t_1, \ldots, s_n \times t_n, \bot \times t_{n+1}, \ldots, \bot \times t_m) \\ \qquad \text{if } n < m, \\ fg(s_1 \times t_1, \ldots, s_n \times t_n, s_{n+1} \times \bot, \ldots, s_m \times \bot) \\ \qquad \text{otherwise.} \end{cases}$$

For instance the product $f(a, g(b)) \times f(f(a, a), b)$ is $ff(af(\bot a, \bot a), gb(b\bot))$. This definition is extended to products of n terms $s_1 \times \ldots \times s_n$ in the obvious way.

Definition 4.1. *A n-ary relation $R \subseteq T_{\mathcal{F}} \times \cdots \times T_{\mathcal{F}}$ is* recognizable *iff the set of all $s_1 \times \ldots \times s_n$ for $\langle s_1, \ldots, s_n \rangle \in R$ is a regular tree language.*

For instance $Id \stackrel{\text{def}}{=} \{(s,s) \mid s \in T_{\mathcal{F}}\}$ is a recognizable relation and an automaton accepting Id needs only one state q (which is final) and rules $ff(q, \ldots, q) \longmapsto q$ for all $f \in \mathcal{F}$. The intersection, the union and the complement of recognizable n-ary relations are also recognizable. The main consequence is that the first-order theory of recognizable relations over finite trees is decidable, or, more precisely:

Theorem 4.2. *Let $\varphi(x_1, \ldots, x_n)$ be a first-order formula involving recognizable relations $R_1, \ldots$ and $Sol(\varphi)$ denote $\{\langle t_1, \ldots, t_n \rangle \mid \models \varphi(t_1, \ldots, t_n)\}$. Then $Sol(\varphi)$ is a recognizable subset of $T_{\mathcal{F}}^n$. Furthermore, from automata $\mathcal{A}_1, \ldots$ recognizing $R_1, \ldots$, one can build an automaton $\mathcal{A}_\varphi$ recognizing $Sol(\varphi)$.*

4.1 Recognizability of the Reachability Relation

We can extend the proof that all $Post^*(X)$ are recognizable languages into a construction showing that the relation $\stackrel{*}{\rightarrow}$ is recognizable, providing a top-down tree automaton for $\stackrel{*}{\rightarrow}$. The main feature of the construction is to add three specific non-terminals I (for identity), R (for rewrite) and R' (for rewrite and termination) and other non-terminals $I_{\perp,s}$, $Q_{X,s}$, $Q_{\perp,s}$, $Q'_{X,s}$ and $Q'_{\perp,s}$ for X, s in $Sub(\Delta)$. The complete automaton is given in the full version of the paper.

Proposition 4.3. *For any Δ, the relation $\stackrel{*}{\rightarrow}$ between PA terms is a recognizable relation, and there is automaton with size $O(|\Delta|)$ recognizing it. The relations $\stackrel{+}{\rightarrow}$, $\rightarrow$ and $\stackrel{w}{\rightarrow}$ for any $w \in Act^*$ are recognizable.*

Since the image and the inverse image of a recognizable language via a recognizable relation is recognizable, the recognizability of $\stackrel{*}{\rightarrow}$ implies the regularity theorems of [LS99] as a byproduct. However, having recognizable $Post^*(L)$ and $Pre^*(L)$ does not necessarily entail the recognizability of $\stackrel{*}{\rightarrow}$. This can be illustrated in the PA framework, as the following remark shows.

Remark 4.4. An alternative definition of the reachability relation for PA is obtained by replacing the rule (R'_{S}) from section 2 with

$$(\mathrm{R}''_{\mathrm{S}}) \qquad \frac{t_1 \stackrel{w_1}{\rightarrow} t'_1 \quad t_2 \stackrel{w_2}{\rightarrow} t'_2}{t_1.t_2 \stackrel{w_1.w_2}{\longrightarrow} t'_2} \text{ if } Const(t'_1) = \varnothing$$

(indeed, why not get rid of these useless terminated processes?). With this new definition, it is still true that, for regular $L \subseteq \mathcal{T}$, $Pre^*(L)$ and $Post^*(L)$ are regular tree languages, but the relation $\stackrel{*}{\rightarrow}$ is in general not recognizable. □

4.2 Costs for Reachability

We can easily write simultaneously a cost grammar associated to the top-down automaton for $\stackrel{*}{\rightarrow}$ which satisfies the following property:

Proposition 4.5. *For any $s, t \in \mathcal{T}$:*

1. $R \overset{*}{\longmapsto} s \times t$ *iff* $s \overset{*}{\to} t$. *Furthermore, if* $s \overset{c}{\to} t$ *then* $c \in C^R$, *and if* $c \in C^R$ *then* $s \overset{c}{\to} t$ *for some* s, t *such that* $R \overset{*}{\longmapsto} s \times t$.
2. $R' \overset{*}{\longmapsto} s \times t$ *iff* $s \overset{*}{\to} t$ *and* t *is terminated. Furthermore, if* $s \overset{c}{\to} t$ *then* $c \in C^{R'}$, *and if* $c \in C^{R'}$ *then* $s \overset{c}{\to} t$ *for some* s, t *such that* $R' \overset{*}{\longmapsto} s \times t$.

5 *TL*, the First-Order Transition Logic

Assume Δ is fixed. The first-order transition logic TL is the first-order logic with process variables $(u, v, \dots)$, the binary predicates $=$, $\overset{*}{\to}$ and $\to$, any other recognizable relation like, for instance, $u \in P$ for P a regular tree language. Observe that whether $t_1, \dots, t_n$ satisfies $\varphi(u_1, \dots, u_n)$ depends on the underlying PA declaration Δ. Theorem 4.2 yields the decidability of TL, or more precisely:

Corollary 5.1. *For any Δ and any TL formula $\varphi(u_1, \dots, u_n)$, we can build an automaton recognizing $Sol(\varphi)$ (a subset of $\mathcal{T}^n$).*

Expressivity Since quantifiers can be used freely, and since equality and other predicates are available, TL is more expressive than the modal logic EF handled in [May99,LS99]. For instance, the confluence of $\overset{*}{\to}$ is expressed by the TL formula $\forall u, v, v' \Big[\big(u \overset{*}{\to} v \wedge u \overset{*}{\to} v'\big) \Rightarrow \exists v'' \big(v \overset{*}{\to} v'' \wedge v' \overset{*}{\to} v''\big) \Big]$.

Furthermore, the logic TL could be extended so that several PA declarations may be used simultaneously. E.g., we can state that the term reachable from X via Δ are the terms reachable via Δ_1 followed by Δ_2 by the formula

$$\forall u \Big[X \overset{*}{\to}_{\Delta} u \;\Leftrightarrow\; \exists v \big(X \overset{*}{\to}_{\Delta_1} v \wedge v \overset{*}{\to}_{\Delta_2} u \big) \Big].$$

Process terms with free variables, e.g., $(X.u) \parallel (v.0)$, can also be used since the predicates encoding the functions symbols are recognizable.

Parameterized verification and model measuring

Computing $Sol(\varphi)$ is more general than deciding validity, satisfiability, or model checking (telling whether $t \models \varphi(u)$ for a given t and φ). In model checking applications, being able to compute $Sol(\varphi)$ under a suitable symbolic representation (a tree automaton $\mathcal{A}_\varphi$ in our case) gives a general approach that smoothly integrate and generalize *parameterized verification* and *model measuring*

Example 5.2. Write X^n for $\overbrace{X \parallel (X \parallel (X \cdots \parallel X) \dots)}^{n \text{ copies of } X}$: the sets $L \overset{\text{def}}{=} \{X^n \mid n = 0, 1, 2, \dots\}$ and $L' \overset{\text{def}}{=} \{Y^n \mid n = 0, 1, 2, \dots\}$ are regular tree languages. Let $\varphi(u, v)$ be some TL formula comparing the behaviors of u and v (e.g. $\varphi \overset{\text{def}}{=} \forall z \big((v \overset{*}{\to} z \wedge z \overset{+}{\to} z) \Rightarrow u \overset{*}{\to} z \big)$, "all loops of v are loops of u"). By computing $Sol(u \in L \wedge v \in L' \wedge \varphi(u, v))$ we can find which values of n make X^n relate to some Y^m (or to all of them) and we can also find which values of m make the property $\varphi(u, Y^m)$ satisfiable (or valid) for the X^n's. $\square$

6 Reachability under (Decomposable) Constraints

In this section, we enrich the basic reachability predicate used in TL and allow to write "decomposable" constraints upon the derivation costs.

6.1 The Decomposable Transition Logic DTL

In [LS99], we proved that reachability via a trace constrained by a regular word language is undecidable but it is decidable for "decomposable" languages (see also [Sch99,Mol96]). Here we define *decomposable cost predicates* along the same lines for unary *cost predicate* P. We write $P(c)$ when P holds for c.

Definition 6.1. *A finite set $\mathcal{DP}$ of cost predicates is a* decomposable family *if*

seq-decompositions: *for all $P \in \mathcal{DP}$ there is a finite index set I and a family $\{P_i^1, P_i^2 \in \mathcal{DP} \mid i \in I\}$ s.t. for all $c, c' \in M$, $P(c \oplus c')$ iff $\bigvee_{i \in I} P_i^1(c) \wedge P_i^2(c')$.*
par-decompositions: *for all $P \in \mathcal{DP}$ there is a finite family $\{P_i^1, P_i^2 \in \mathcal{DP} \mid i \in I\}$ s.t. for all $c, c' \in M$, $P(c \otimes c')$ iff $\bigvee_{i \in I} P_i^1(c) \wedge P_i^2(c')$.*
unit-decompositions: *for all $P \in \mathcal{DP}$ and all costs c appearing in Δ, there is a finite family $\{P_i^c \in \mathcal{DP} \mid i \in I\}$ s. t. for all $c' \in M$, $P(c \oplus c')$ iff $\bigvee_{i \in I} P_i^c(c')$.*

A predicate P is *decomposable* if it belongs to a decomposable family. DTL (Decomposable Transition Logic) is the first-order logic that extends TL by allowing all atoms $u \xrightarrow{\exists c\ P(c)} v$ where P is any decomposable predicate.

$u \xrightarrow{\exists c\ P(c)} v$ is short for "$\exists c, u \xrightarrow{c} v \wedge P(c)$" and holds iff there is a derivation $u \xrightarrow{c} v$ such that $P(c)$ holds. Observe that we require that any cost variable be immediately quantified upon (like the freeze quantification of [AH94]), hence the cost variable c is always bound and we sometimes simply write $u \xrightarrow{\exists P} v$.

Using the tree automata approach, we can show that the relation $\xrightarrow{\exists P}$ is recognizable for any decomposable predicate P, which yields the decidability theorem:

Theorem 6.2. *The logic DTL is decidable.*

We now look at instances of DTL where costs measure some form of timing.

6.2 The Timed Transition Logic *TTL*

TTL (Timed Transition Logic) is the first-order logic that extends TL by allowing all atoms $u \xrightarrow{\exists \tau} v$ where τ is a *time constraint* built according to the grammar:

$$\tau ::= c < C \mid \neg\tau \mid \tau \wedge \tau$$

where the C's can be any numerical constant from a time domain $\mathbb{T}$ that can be $\mathbb{N}$, or $\mathbb{Q}^+$, or $\mathbb{R}^+$ (and where c is the free cost variable of τ). Since these time constraints can be expressed by decomposable predicates, we have

Proposition 6.3. *For any time constraint τ, the relation $s \xrightarrow{\exists \tau} t$ is recognizable.*

allowing the following instantiation of Theorem 6.2

Theorem 6.4. *The logic TTL is decidable.*

TTL can be enriched with the predicate $u \xrightarrow{Unbounded} v$, meaning that going from u to v may take arbitrarily long time:

Lemma 6.5. *The relation $\xrightarrow{Unbounded}$ is recognizable.*

7 The Transition Logic with Constraints TLC

In this section, we add a first-order logic of costs to the transition logic. The resulting two-sorted logic, called TLC, is very expressive and allows parameterized verification with parameters ranging over costs rather than "processes". For TLC, costs are Parikh costs and the cost $c = (n_1, \dots, n_p)$ of a derivation $s \xrightarrow{*} t$ records the number of occurrences of each action of Act along the derivation. Since c is now a p-tuple of integers, we often write $x_1, \dots, x_p$ or $\bar{x}$ instead of c. The formula over costs are Presburger formula. This allows to state properties as "$s \xrightarrow{w} t$ with *as many actions a as actions b*".

TLC allows two kinds of atoms: all $R(u_1, \dots, u_n)$ for R a recognizable relation as in TL and all $u \xrightarrow{\exists \bar{x} \psi(\bar{x}, \bar{y})} v$ where $\psi(\bar{x}, \bar{y})$ is a Presburger formula whose free variables are partitioned into $\bar{x}$ a tuple of p integer variables, for the cost of the derivation and the arbitrary parameters $\bar{y}$. $u \xrightarrow{\exists \bar{x} \psi(\bar{x}, \bar{y})} v$ is short for "$\exists c, u \xrightarrow{c} v \wedge \psi(c, \bar{y})$". Observe that only $u, v, \bar{y}$ are free in $u \xrightarrow{\exists \bar{x} \psi(\bar{x}, \bar{y})} v$. The negation of $u \xrightarrow{\exists \bar{x} \psi(\bar{x}, \bar{y})} v$ can be written $\forall \bar{x} \big(u \xrightarrow{\bar{x}} v \Rightarrow \psi'(\bar{x}, \bar{y})\big)$ where ψ' is $\neg\psi$, another Presburger formula, and we shall use this notation freely. TLC formulas are given by the abstract syntax:

$$\varphi ::= Atom \mid \varphi \wedge \varphi \mid \neg\varphi \mid \exists u \varphi \mid \exists y \varphi$$

Since the full TLC is undecidable (Prop. 7.1) we introduce two fragments that will be shown decidable (Theo. 7.3):

the parameterized existential fragment: which is the set of closed formulas that can be written under the form $(\exists | \forall\ \bar{y})^*(\exists\ u)^*[\vee \wedge Atoms]$, and

the parameterized universal fragment: which is the set of closed formulas that can be written under the form $(\exists | \forall\ \bar{y})^*(\forall\ u)^*[\vee \wedge \neg Atoms]$.

The restriction on the polarity of atoms only applies only to reachability atoms "$u \xrightarrow{\exists \bar{x} \psi(\bar{x}, \bar{y})} v$" with some non-empty $\bar{y}$ hence $u \xrightarrow{*} v$, etc., can be negated freely. Observe that one fragment only contains formulas that are (equivalent to) the negation of a formula of the other fragment.

7.1 The Difference between TLC and Temporal Logics

Temporal logics do not have a mechanism for identifying states precisely and relate them. On the other hand, they can refer to a given path, state properties

that hold along this path, and relate paths. When we write $u \xrightarrow{\exists \bar{x} \psi(\bar{x},\bar{y})} v$ in TLC, we state that u may go to v via a path having some cost properties, but we cannot isolate this path and refer to it again. Additionally, temporal modalities are recursive by nature, while transition logics only have the fixpoint built-in $\xrightarrow{*}$. As a consequence, simple temporal modalities like EF can be expressed in the transition logic but more complex constructions like E_U cannot.

The ability to refer to states is the specific feature of transition logics: these logics can distinguish bisimilar processes. And the rich language for constraints allows to express many counting properties that occur naturally in verification.

7.2 Expressing Properties with *TLC*

We consider the communication protocol example used in [BEH95b], where they focus on the two actions *req* (for requests) and *ack* (for acknowledgments).

TLC can state that, between an initial state and a final state, a protocol sends as many acknowledgments *ack* as requests *req* :

$$\forall u, v \ (u \in Init \wedge v \in Final) \Rightarrow \big(\forall x_{ack}, x_{req}((u \xrightarrow{x_{ack},x_{req}} v) \Rightarrow x_{ack} = x_{req}\big)$$

where "$x_{ack} = x_{req}$" is the $\psi(\bar{x})$ constraint. (We assumed that $Init$ and $Final$ are regular tree languages.)

TLC can also state that at any position along a path from an initial state to a final state, the number of sent requests is always greater than or equal to the number of received acknowledgments. This is specified as follows:

$$\forall u, v, w \left(\begin{array}{c} u \in Init \wedge v \in Final \wedge u \xrightarrow{*} w \wedge w \xrightarrow{*} v \\ \Rightarrow \forall x_{ack}, x_{req}\big(u \xrightarrow{x_{ack},x_{req}} w \Rightarrow x_{req} \geq x_{ack}\big) \end{array} \right)$$

TLC can further impose that all traces belong to $req^* ack^*$. We add to the previous formula the formula stating that if a state is reached by emitting an acknowledgment, then all paths from this state to a final state contain no request.

$$\forall u, v, w \ \left(\begin{array}{c} u \in Init \wedge v \in Final \wedge (u \xrightarrow{\exists x_{ack}, x_{ack}=1} w) \\ \Rightarrow \forall x_{ack}, x_{req}\big(w \xrightarrow{x_{ack},x_{req}} v \Rightarrow x_{req} = 0\big) \end{array} \right)$$

Observe that these three examples are all in the parameterized universal fragment of TLC and could be enriched by using parameters.

7.3 Decidability Issues for *TLC*

The first result is a negative one.

Proposition 7.1. *TLC is undecidable, even when restricted to the fragment without parameters and with only $\exists\forall$ quantification for process variables.*

This result prompted the introduction of the parameterized fragments of TLC. Let $\Phi \equiv (\exists|\forall\ \bar{y})^*(\exists\ u)^*\varphi$ be a parameterized existential TLC formula. Let $\phi(y_1, \ldots, y_k)$ denote the "$(\exists\ u)^*\varphi$" part and $Sol(\phi) = \{\langle n_1, \ldots, n_k\rangle \mid \models \phi(n_1, \ldots, n_k)\}$.

Theorem 7.2. *$Sol(\phi)$ is an effectively computable semilinear set.*

The proof of this result uses an extension of tree automata which combines a cost grammar with the usual tree automata rules. This class of automata is closed under product and union and the reachability of a state is decidable. Moreover in our case, the set of costs associated to a state is an effectively computable semilinear set and the relation $\xrightarrow{c}$ is accepted by an automaton with cost.

Theorem 7.3. *The parameterized existential and the parameterized universal fragments of TLC are decidable.*

8 Conclusion

The recognizability of $\xrightarrow{*}$ extends our earlier results on reachability sets. This also opens new directions for automata-theoretic approaches to the verification of PA-processes, since being able to compute the set of solutions of a transition logic formula allows a smooth and general approach to the verification of parameterized properties for parameterized systems. Additionally, the automata-theoretic approach relies on quite simple constructions. The consequence is that we can easily extend it in various ways, as we demonstrated with reachability under decomposable cost predicates, with various timed extensions of the transition logic, and with TLC where both PA-processes and Parikh costs can be constrained via parameterized formulas. An important goal for future work is to analyze the computational complexity of the various ideas we proposed. This should help understand what cost sets and what decomposable predicates can be handled in practice, and what restrictions may be fruitfully imposed on transition logics so that they remain computationally tractable.

References

AELP99. R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric temporal logic for "model measuring". In *Proc. 26th Int. Coll. Automata, Languages, and Programming*, vol. 1644 of L.N.C.S., pages 159-168. Springer, 1999.

AH94. R. Alur and T. A. Henzinger. A really temporal logic. *J. of the ACM*, 41(1):181-203, 1994.

BEH95a. A. Bouajjani, R. Echahed, and P. Habermehl. On the verification problem of nonregular properties for nonregular processes. In *Proc. 10th IEEE Symp. Logic Comp. Science*, pages 123-133, 1995.

BEH95b. A. Bouajjani, R. Echahed, and P. Habermehl. Verifying infinite state processes with sequential and parallel composition. In *Proc. 22nd ACM Symp. Princ. of Programming Languages*, pages 95-106, 1995.

BW90. J. C. M. Baeten and W. P. Weijland. *Process Algebra*, vol. 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge Univ. Press, 1990.

CDG+99. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications, 1997-99. Available at http://www.grappa.univ-lille3.fr/tata.

CJ99. H. Comon and Y. Jurski. Timed automata and the theory of real numbers. In *Proc. 10th Int. Conf. Concurrency Theory*, vol. 1664 of L.N.C.S., pages 242-257. Springer, 1999.

DT90. M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th IEEE Symp. Logic in Computer Science*, pages 242-248, 1990.

EK99. J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural dataflow analysis. In *Proc. 2nd Int. Conf. Found. of Soft. Sci. and Comp. Struct.*, vol. 1578 of L.N.C.S., pages 14-30. Springer, 1999.

EP00. J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *Proc. 27th ACM Symp. Principles of Programming Languages*, 2000.

Esp97. J. Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundamenta Informaticae*, 31(1):13-25, 1997.

ET99. E. A. Emerson and R. J. Treer. Parametric quantitative temporal reasoning. In *Proc. 14th IEEE Symp. Logic in Comp. Sci.*, pp 336-343, 1999.

HJ99. Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *Proc. 26th Int. Coll. Automata, Languages, and Programming*, vol. 1644 of L.N.C.S., pages 412-421. Springer, 1999.

JKM98. P. Jançar, A. Kučera, and R. Mayr. Deciding bisimulation-like equivalences with finite-state 201 449 243 461 processes. In *Proc. 25th Int. Coll. Automata, Languages, and Programming*, vol. 1443 of L.N.C.S., pages 200-211. Springer, 1998.

Kuč96. A. Kučera. Regularity is decidable for normed PA processes in polynomial time. In *Proc. 16th Conf. Found. of Software Technology and Theor. Comp. Sci.*, vol. 1180 of L.N.C.S., pages 111-122. Springer, 1996.

Kuč97. A. Kučera. How to parallelize sequential processes. In *Proc. 8th Int. Conf. Concurrency Theory*, vol. 1243 of L.N.C. S., pages 302-316. Springer, 1997.

LS99. D. Lugiez and Ph. Schnoebelen. The regular viewpoint on PA-processes. September 1999. To appear in *Theor. Comp. Sci.*

May97. R. Mayr. Tableaux methods for PA-processes. In *Proc. TABLEAUX'97*, vol. 1227 of L.N.A.I., pages 276-290. Springer, 1997.

May99. R. Mayr. Decidability of model checking with the temporal logic EF. May 1999. To appear in *Theor. Comp. Sci.*

May00. R. Mayr. Process rewrite systems. Information and Computation, 156(1/2):264-286, 2000.

Mol96. F. Moller. Infinite results. In *Proc. 7th Int. Conf. Concurrency Theory*, vol. 1119 of L.N.C.S., pages 195-216. Springer, 1996.

Par66. R. J. Parikh. On context-free languages. *J.A.C.M*, 13(4):570-581, 1966.

Plo81. G. D. Plotkin. A structural approach to operational semantics. Lect. Notes, Aarhus University, Aarhus, DK, 1981.

Sch99. Ph. Schnoebelen. Decomposable regular languages and the shuffle operator. *EATCS Bull.*, 67:283-289, 1999.

Sei94. H. Seidl. Finite tree automata with cost function. *Theoretical Computer Science*, 126(1):113-142, 1994.

Non Interference for the Analysis of Cryptographic Protocols *

Riccardo Focardi[1], Roberto Gorrieri[2], and Fabio Martinelli[3]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia, Italy.
focardi@dsi.unive.it
[2] Dipartimento di Scienze dell'Informazione, Università di Bologna, Italy.
gorrieri@cs.unibo.it
[3] Istituto per le Applicazioni Telematiche C.N.R., Pisa, Italy.
Fabio.Martinelli@iat.cnr.it

Abstract. Many security properties of cryptographic protocols can be all seen as specific instances of a general property, we called Non Deducibility on Composition (NDC), that we proposed a few years ago for studying information flow properties in computer systems. The advantage of our unifying theory is that formal comparison among these properties is now easier and that the full generality of NDC has helped us in finding a few new attacks on cryptographic protocols.

1 Introduction

Many security properties of cryptographic protocols have been identified in recent years, such as *secrecy* (confidential information should be available only to the partners of the communication), *authentication* (capability of identifying the other partner engaged in a communication), *integrity* (assurance of no alteration of message content), *non repudiation* (assurance that a signed document cannot be repudiated by the signer), *fairness* (in a contract, no party can obtain advantage by ending the protocol first), and some others.

Even if there is a widespread agreement on what is the intended meaning of these properties, under a closer scrutiny one realizes that they are very slippery properties, especially authentication. As a matter of fact, formal definitions, e.g. of authentication, have rarely been given, not widely agreed upon, usually not compared and only recently proposed in the literature (see, e.g., [5,17,21,26]). This is sometimes due to the fact that we first need a formal model on which the problem is defined (and this is often a source of possible proliferation of different proposals) and then a formal definition w.r.t. the chosen model. Moreover, even when a formal definition is given, usually this is not (easily) comparable to others, due to different mathematical assumptions of the model.

* Work partially supported by MURST Progetto TOSCA and Progetto "Certificazione automatica di programmi mediante interpretazione astratta"; CNR Progetto "Modelli e Metodi per la Matematica e l'Ingegneria"; CSP Progetto "ISA: Isp Secured trAnsactions".

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 354–372, 2000.
© Springer-Verlag Berlin Heidelberg 2000

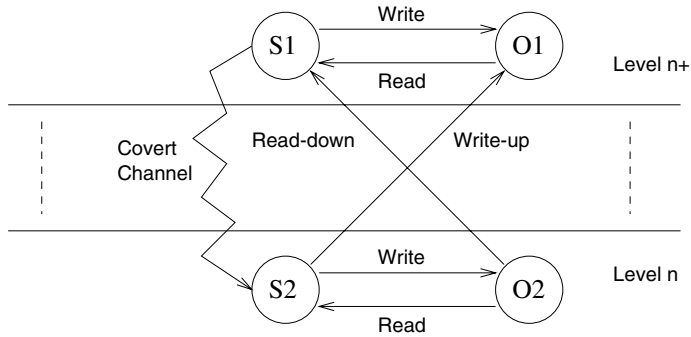


Fig. 1. Multilevel security: a high subject S_1 cannot write a low object O_2 and a low subject S_2 cannot read a high object O_1.

Our claim is that a classic approach to security, used to study information flow in multilevel [4] computer systems, can be profitably used also for the analysis of security properties in network protocols.

1.1 Multilevel Security and non Interference

In a multilevel systems, processes/users and objects are bound to a specific security level (e.g., in the military jargon, unclassified, classified, secret and top secret) and information can only flow from low levels to higher ones. This is usually implemented by constraining the possible actions of processes according to the rules of *no read-up* and *no write-down* (see Fig. 1).

The advantage of this approach w.r.t. conventional approaches used in commercially available operating systems (e.g., Unix) is that the possible information disclosures caused by the inadvertent execution of a Trojan Horse program is confined inside the level of the user that executed it. However, these two rules are not enough as indirect information flows, usually called *covert channels*, may be possible when using some shared resource. For instance, it is not difficult to build a Trojan Horse program that, once executed by a high level user, is able to downgrade information by synchronizing with a low level process on the system side-effects generated by repeatedly filling the shared hard disk: at a predefined initial time, the high process can trasmit a bit 0 by causing a disk-full error on the low level attempt to write, or a bit 1 by allowing the low process to write, hence one bit each two clock cycles.

To solve the problem of preventing unauthorized information flows, be they direct or indirect, in the last two decades many proposals have been presented, starting from the seminal idea of *non interference* proposed in [16] for deterministic state machines. In recent work [10,11], two of the authors have studied the many non interference-like definitions in the literature, by defining all of them uniformly in a common process algebraic setting, producing the first taxonomy of these security properties reported in the literature.

In [10,11], we use a CCS-like process algebra [23], called Security Process Algebra (SPA for short), where the set of actions is partitioned into two sets L and H of low actions and high ones, respectively. Processes built by using only actions in H (L) are by construction high (low) level processes. These processes are secure because their activities (expressed by the actions they perform) are confined inside the high (low) level. More interesting is the case of mixed (with actions from both L and H) processes, i.e., of those processes that even if belonging to the high (low) level may perform interactions with low (high) objects, because for them we want to know if they allow information to flow in the wrong direction. Among the many non interference-like properties, we advocate one special property, called *Non Deducibility on Composition* (NDC for short), that can be expressed as follows:

$$E \in NDC \text{ iff } \forall \Pi \in \mathcal{E}_H : (E \parallel \Pi)\backslash H \approx E\backslash H$$

where $\mathcal{E}_H$ is the set of all high level processes, $\approx$ is a behavioural equivalence relation,[1] $\parallel$ is the CCS parallel composition and $\backslash$ is the CCS restriction operator. Hence, on the one hand, $E\backslash H$ is able to exhibit only the low level behaviour of E, while $(E \parallel \Pi)\backslash H$ is the low level behaviour of $E \parallel \Pi$. The basic intuition is that the requirement of

No information flow from high to low

is expressed by

No high level process can change the low behaviour.

1.2 Non Interference for Security Protocols

NDC essentially says that, given two groups of users H and L, there is no information flow from H to L iff there is no way for H to modify the behaviour of L. Analogously, we may think that L is the set of the honest participants to a protocol and H is the external, possibly malicious, environment, i.e. the set of possible intruders (or enemies). Following the analogy, no information flow from high to low means that the intruders have no way to change the low behaviour of the protocol.

To set up this correspondence more precisely, we have to single out the high level actions and the low level ones in this setting. We should assume that an intruder may have complete control of the network, and so it is reasonable to assume that the public channels (i.e., the names used for message exchange) are the high level actions. On the other hand, as a protocol specification is usually completely given by message exchanges, it is not clear what are the low level actions. In our approach, the low level actions are extra observable actions that are included into the protocol specification to observe properties of the protocol. Of course, the choice of these extra actions is property dependent. For instance,

[1] Actually, NDC in [10,11] is this property when $\approx$ is trace equivalence; other similar properties have been proposed by changing the relation, e.g., $BNDC$ is as above where $\approx$ is weak bisimulation.

we will see that to model some form of authentication as in [20], it is enough to include special start/commit actions for all the honest participants.

Furthermore, enemies should not be allowed to know secret information in advance: as we assume *perfect cryptography* (a crypted information can be known by an enemy only if he knows the decryption key), the initial knowledge of an ebemy must be limited to include only publicly available information, such as names of entities and public keys, and its own private data (e.g., enemy's private key). Hence, by following [15,14,13], the set $\mathcal{E}_C^{\phi_I}$ of all the possible high level processes is as follows: $\mathcal{E}_C^{\phi_I} = \{X \mid sort(X) \subseteq C \text{ and } ID(X) \subseteq \mathcal{D}(\phi_I)\}$, where C is the set of public channel names, $ID(X)$ is the set of messages that syntactically appear in X, ϕ_I is the initial knowledge given to any enemy X, and $\mathcal{D}$ is a deduction system that manipulates (blocks of) messages in the obvious way (e.g., a crypted information can be disclosed if the decryption key is known). By requiring that all the messages in $ID(X)$ are deducible from ϕ_I we are stating that the enemy cannot know in advance messages that are not explicitly given. The NDC property for a protocol P can hence be reformulated as:

$$P \in NDC \text{ iff } \forall X \in \mathcal{E}_C^{\phi_I} : (P \parallel X) \setminus C \approx P \setminus C$$

On the one hand, $P \setminus C$ represents the secure specification of the protocol P running in isolation on perfectly secure channels. The visible behaviour of P is given by the property dependent, extra observables included in the specification. Hence, the behaviour of $P \setminus C$ should describe the security property of interest. On the other hand, if $P \setminus C$ is equivalent to $(P \parallel X) \setminus C$, then this clearly means that X is not able to modify in any way the observable execution of P, i.e., the security property hold.

The actual scheme we use, called $GNDC$, is a bit more general, where $\approx$ is any pre-order and $P \setminus C$ is replaced by a function on P, $\alpha(P)$, expressing the property as a set of processes having a special format that more directly and intutively recall the property of interest. Nonetheless, it is possible to show that NDC is the strongest property we can reasonably define over cryptographic protocols (for more details, see [15]).

Interestingly enough, when the observational equivalence $\approx$ is trace equivalence (two systems are equivalent if they perform the same set of traces), then NDC can be characterized in a simpler way, by finding a canonical, most general enemy that can be used in place of all (see [15]). By removing the universal quantification, NDC can be verified by one single, albeit huge, check. The most general intruder is an intruder that can eavesdrop/intercept any message (adding the intercepted information to its knowledge set), as well as produce new messages with pieces of information he knows.

1.3 Plan of the Paper

What do we want to show with this paper? Our primary goal is to substantiate our claim that most (maybe all) security properties proposed for the analysis of cryptographic protocols are expressible as suitable instances of the $GNDC$

schema above, by suitably choosing the property dependent, extra observable actions as well as a suitable behavioural equivalence. Some work in this direction has been reported in [14,13,15]. We think that the advantages of our approach include at least the following:

- *one check for all*: As all the properties are defined in the same NDC style, it is possible to put in the specification the extra actions for all the properties of interest, hence obtaining that one check for this rich case implies that all the properties are satisfied.
- *formal comparison*: as the definitions are now given in a uniform style, it should be easier to compare the relative merits; this is especially true for slippery properties such as the many varieties of authentication (e.g., see [13,15] for some preliminary results in this direction).
- *accuracy*: So far we have analyzed about 40 protocols (with the help of an automatics tool [8,9]) of a well-known library of crypto-protocols [6]. Two supposedly correct protocols have been shown incorrect and for a few additional flawed protocols some new attacks have been found. Our experience hence supports our claim that a protocol passing the NDC test is more likely to be flaw free.

The paper is organized as follows. In Section 2 we gently introduce the reader to the realm of security properties; by means of some simple examples, five security properties are informally described. Section 3 shows that all these properties are actually instances of the general scheme $GNDC$ and that NDC (when all the suitable extra actions have been inserted) implies them all. Section 4 shows one larger example, the Woo & Lam mutual authentication protocol as reported in Schneier's textbook [27]. This version of the protocol is flawed. Finally, Section 5 reports some final remarks and future work.

2 Security Properties

In this section we present some typical security properties through some simple examples. All these properties have been defined for different aims and have been formalized using various models. Indeed, the examples will allow us to identify a *general common idea* behind all of these properties.

2.1 A Simple Key-Exchange Protocol

The first example we consider is a simple key-exchange protocol (see [11]) with public key cryptography. There is a process KDC (Key Distribution Center) on a remote host which is devoted to the distribution of the public keys. In particular, when *Alice* (A) need to send some secret information to *Bob* (B), A asks KDC for B's public key. Then, A can encode every secret message with this key and is guaranteed that only B will be able to read it. In particular, in the protocol we are going to analyze, A sends to B a session key encoded with the

public key of B. Such session key will be used for every further communication, until the two users decide to establish a new session with a new session key.

The aim of the protocol is simply to distribute to B the session key generated by A. Since A uses the public key of B, she is assured that only B will know the session key and so there is a form of implicit one-side authentication of B for A. The opposite authentication is not valid since B has no guarantees about who he is talking to. Anyone can encrypt a session key with B's public key and send it pretending to be A. So, B will presumably check the identity of A during the session (if this is required). We could imagine a situation of a remote login on a machine B by a user A. First the user establishes a session key with the remote login server using the protocol above. Then the communication proceeds encrypted with the session key and the server checks the identity of A using, for instance, some login/password mechanism. However A is assured to communicate with B since only B can have received the session key.

In order to formalize the protocol we use the notation $A \rightarrow B : msg$ representing the sending of message msg from A to B. A possible definition of the protocol could be the following sequence of four message exchanges:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow KDC & : \; A, B \\ \text{Message 2} & KDC \rightarrow A & : \; PK_B \\ \text{Message 3} & A \rightarrow B & : \; \{K_{sess}\}_{PK_B} \\ \text{Message 4} & B \rightarrow A & : \; \{M\}_{K_{sess}} \end{array}$$

where Message 1 is the request, sent from A to KDC, of B's public key; Message 2 is the reply from KDC to A, containing the public key PK_B of B; Message 3 contains the session key K_{sess} encoded using the public key PK_B and is sent to B; finally, in Message 4, B uses the session key to send a message M to A.

Different security properties can be now considered: (*i*) *message authenticity*, e.g., message M should be authentic from B (or as it would have been sent from B) since only A and B should know the session key at the end of the protocol; (*ii*) *entity authentication*, e.g., if A receives the last message encrypted with the correct key, then A should be guaranteed that B has run the protocol with her (or at least is "alive"); (*iii*) *secrecy*, e.g., at the end of the protocol, the session key and the message M should be known only to A and B. In the following, we consider all of these properties, using the protocol above as a running example.

2.2 Message Authenticity

Suppose that we can fix the message M that B is willing to send to A. This means that M does not represent a generic message, but it is a particular one. If we can do this, we can check message authenticity by just considering all the possible runs of the protocol and by requiring that, in such runs, A always receives the correct message M, i.e. the message B wanted to send to A. If this is true, we can conclude that the protocol is indeed guaranteeing that no one is able

to force A accepting a faked message M'. This notion of message authenticity is due to Abadi and Gordon [2]. [2]

A first important thing to observe is that considering all the possible runs is not enough. At least we need to be more precise about what we mean by run. As a matter of fact, we have to consider all the possible executions of the protocol in every possible (potentially hostile) environment. It is certainly different if we consider the protocol execution with or without the presence of some malicious enemy which tries to send a faked M'. (As mentioned in the Introduction, we have to consider an initial knowledge ϕ_I but, for sake of readibility, we often omit it.) We can thus rephrase the message authenticity property as follows:

> "Whatever hostile environment is considered, A will never receive (as part of Message 4) during all her possible runs a message different from M".

A second important issue is now also evident. We are requiring that a particular piece of information sent inside a particular message differs from a certain fixed message M. Indeed, this property looks really ad-hoc for the key exchanged protocol considered here. This is quite typical when trying to define precisely security properties, since they often depend on the structure of the analyzed protocol/system. We can make more intuitive the specification by using an event $received(m)$ corresponding to the fact that A is receiving message m. If $P(m)$ is the protocol where *Bob* is willing to send message m to *Alice* we can just state that

> "$P(M)$ guarantees message authenticity if whatever hostile environment is considered, an event $received(M')$ with $M' \neq M$ can never occur".

We will further generalize this idea in the following. Before that we show that the protocol we have considered until now does not guarantee message authenticity. The weakness is indeed in the second message. An enemy can easily intercept the public key of B and substitute it with its own key. This allows the enemy to learn the session key and to send a faked message M' as follows:[3]

Message 1	$A \rightarrow KDC$	: A, B	
Message 2	$KDC \rightarrow \mathbf{E}(A)$	: PK_B	E intercepts this
Message 2′	$\mathbf{E}(KDC) \rightarrow A$	: $\mathbf{PK_E}$	
Message 3	$A \rightarrow \mathbf{E}(B)$	: $\{K_{sess}\}_{\mathbf{PK_E}}$	
Message 4	$\mathbf{E}(B) \rightarrow A$	: $\{\mathbf{M'}\}_{K_{sess}}$	event $received(M')$

[2] Indeed, in [2] a universal quantification over all the possible messages to be sent is required. For the sake of simplicity we do not consider it here.

[3] We denote with $E(U)$ the enemy which is impersonating the entity U. So $E(U) \rightarrow A : M$ means that E sends M to A by simulating U, while $A \rightarrow E(U) : M$ that E intercepts the message M from A to U (so U receives nothing). However we must point out that this message sequences notation should be used only for the intuitive description of the protocols (and attacks) and not for their formal analysis as remarked by many authors (e.g., see [1]).

Thus, message authenticity does not hold since, in this particular execution, an event $received(M')$ occurs and M' can be whatever message (different from M) the enemy is willing to send to A.

2.3 Entity Authentication

We now consider another security property: entity authentication. This property is more subtle. Informally, entity authentication should allow the verification of an entity's claimed identity, by another entity. There are several attempts in the literature to formalize this notion. Here, we follow the ones based on *correspondence* between actions of the participants (e.g., see [18,21,30]).

As an example, in our protocol we would like that whenever A receives the last message then B has indeed executed the protocol. Consider two events $commit(A, B)$ and $run(B, A)$ representing the fact that A has successfully terminated the protocol apparently with B and B has at least started the protocol (i.e., he has received a session key) with A. It is now sufficient to require that event $commit(A, B)$ is always preceded by event $run(B, A)$ [21]. In other words $commit(A, B)$ should not happen if B has not started the protocol. Similarly to the previous property we can require that:

> "P guarantees entity authentication of B with respect to A if whatever hostile environment is considered, it can never occur an event $commit(A, B)$ when $run(B, A)$ has not occurred previously".

Note that the same attack considered for message authenticity is also an attack for entity authentication:

Message 1	$A \rightarrow KDC$:	A, B	
Message 2	$KDC \rightarrow \mathbf{E}(A)$:	PK_B	E intercepts this
Message 2′	$\mathbf{E}(KDC) \rightarrow A$:	$\mathbf{PK_E}$	
Message 3	$A \rightarrow \mathbf{E}(B)$:	$\{K_{sess}\}_{\mathbf{PK_E}}$	
Message 4	$\mathbf{E}(B) \rightarrow A$:	$\{\mathbf{M}'\}_{K_{sess}}$	event $commit(A, B)$

Since B is doing nothing, no event $run(B, A)$ can happen and the entity authentication property does not hold. As a matter of fact, in the attack sequence the enemy is indeed able to mask as B. This means that A cannot be sure about the identity of the other party, i.e., no entity authentication is guaranteed.

Note that entity authentication and message authenticity are indeed different properties. As an example, if we do not have a message "to be sent" by the entity that we want to authenticate, message authenticity property becomes useless. Consider the following (faulty) authentication protocol:

Message 1	$A \rightarrow B$:	$\{N_A\}_{K_{AB}}$
Message 2	$B \rightarrow A$:	N_A

In order to verify the identity of B, A sends a challenge N_A (typically a random number) to B encrypted with symmetric key K_{AB} which is only known by A

and B. Only B will be able to decrypt N_A and send it back to A. Note that here B has no private messages to send to A. As a consequence, the following entity authentication attack does not represent a message authenticity attack:

$$\begin{array}{llll}
\text{Message 1} & A \to \mathbf{E}(B) : & \{N_A\}_{K_{AB}} & \\
\text{Message 1}' & \mathbf{E}(B) \to A : & \{N_A\}_{K_{AB}} & \\
\text{Message 2}' & A \to \mathbf{E}(B) : & N_A & \\
\text{Message 2} & \mathbf{E}(B) \to A : & N_A & \text{events } \mathit{commit}(A,B), \\
 & & & \quad \mathit{received}(N_A)
\end{array}$$

This is a typical parallel session attack: the enemy intercepts the first message and starts a new session of the protocol with A. Basically, the enemy uses this second session to obtain from A the value N_A. Finally the enemy can conclude the first session successfully masking as B. Note that, again, we have a $\mathit{commit}(A, B)$ event with no $\mathit{run}(B, A)$. Note also that we have tried to detect a possible message authenticity attack by observing the $\mathit{received}(N_A)$ event. However N_A is exactly the expected message and, moreover, no other message would be accepted by A. In other words, we have no message here to authenticate and entity authentication should be guaranteed (indeed it is not) by the possibility for B of decrypting a message.

2.4 Secrecy

Let us now consider the third property: secrecy. This is quite intuitive and requires that messages declared to be secret should not be learnt by unauthorized users. We can consider a new event $\mathit{learnt}(M)$ that represents the fact that a certain (secret) message M has been learnt by the external environment (i.e., by the enemy). So, this new (low) event is performed by the enemy and not by the honest participants, as for the previously analysed properties. In our first example of attack we had such an event for K_{sess}:

$$\begin{array}{llll}
\text{Message 1} & A \to KDC : & A, B & \\
\text{Message 2} & KDC \to \mathbf{E}(A) : & PK_B & \text{E intercepts this} \\
\text{Message 2}' & \mathbf{E}(KDC) \to A : & \mathbf{PK_E} & \\
\text{Message 3} & A \to \mathbf{E}(B) : & \{K_{sess}\}_{\mathbf{PK_E}} & \text{event } \mathit{learnt}(\mathbf{K_{sess}}) \\
\text{Message 4} & \mathbf{E}(B) \to A : & \{\mathbf{M}'\}_{\mathbf{K_{sess}}} &
\end{array}$$

Indeed, secret key K_{sess} is learnt by the enemy when it is sent encrypted with enemy's public key. We can thus formulate secrecy in our usual style as follows:

> "P guarantees secrecy of m if whatever hostile environment is considered, the event $\mathit{learnt}(m)$ can never occur".

2.5 The Example (Partially) Repaired

In this section we show how to repair the initial protocol in order to avoid the attacks reported above (even though other ones are possible). The attacks work

since it is possible to fake Message 2 in order to provide a wrong public key, i.e., not the one associated to B. In actual implementations, the KDC sends the public key together with the name of the associated user, all signed with its own key. Hence, the resulting protocol is the following:

$$\begin{array}{lll} \text{Message 1} & A \rightarrow KDC: & A, B \\ \text{Message 2} & KDC \rightarrow A & : \{PK_B, B\}_{SK_{KDC}} \\ \text{Message 3} & A \rightarrow B & : \{K_{sess}\}_{PK_B} \\ \text{Message 4} & B \rightarrow A & : \{M\}_{K_{sess}} \end{array}$$

where Message 2 now consists of the pair (PK_B, B) encrypted with the private key of KDC (SK_{KDC}), in such a way that everyone can decrypt it and be sure that the associated public key PK_B has been originated by KDC. It is easy to see that the attacks previously shown are not possible anymore, since the enemy is not able to generate the block $\{PK_E, B\}_{SK_{KDC}}$.

This updated protocol may still be subject to some form of attacks, namely replay attacks. These attacks work since the enemy is able to re-use the information obtained in previous runs of the protocol in order to fake messages for A. Imagine the situation where the session key K_{sess} between A and B has been safely established and is used to send two different messages, say M, M', to A, by simply adding another step to the previous protocol:

$$\text{Message 5} \quad B \rightarrow A: \ \{M'\}_{K_{sess}}$$

The enemy could eavesdrop Message 4, intercept Message 5 and replay the Message 4 to A. Thus, A receives two times message M! For example, if M represents a bank transfer request this attack could result in a double transfer of money. This is a message authenticity attack and is revealed since we obtain two events $received(M)$ instead of $received(M)$, $received(M')$.

This kind of attacks may be prevented by inserting freshness in the messages (such as newly generated random numbers). It is not our intention to give here a complete spectrum of attacks on cryptographic protocols; we just show that these are very subtle and make the design of such protocols very challenging. (For some guidelines about cryptographic protocols design see [3].)

2.6 Non Repudiation and Fairness

There are other interesting properties that can be rephrased in our common style. For example, *non repudiation* and *fairness* (i.e., fair message exchange). The former is related to the possibility of considering a certain message as a signed contract, which is thus not repudiable by the sender. This is typically obtained through protocols which are based on some electronic signature mechanism. Non repudiation is of course very important for electronic commerce. The latter property requires that mutual information exchanges are performed in a fair way, i.e., no one of the parties involved in the exchange should get an advantage by obtaining the information before the other party is also able to obtain it. Indeed, if an exchange is not fair, the advantaged party could refuse to give

its information once it has received the information from the other party. This property is sometimes required together with non repudiation when the parties want to simultaneously exchange two non repudiable messages (for example, one message could be the electronic payment and the other one could be the corresponding electronic receipt). Here we follow the treatment in [15,25].

(Fair) Non repudiation protocols are often quite complex. Here, in order to illustrate these properties we consider the following very simple example (see also [31]):

$$\begin{array}{ll} \text{Message 1} & A \rightarrow B: \ M, \{M, B\}_{SK_A} \\ \text{Message 2} & B \rightarrow A: \ \{M, A\}_{SK_B} \end{array}$$

A sends a message M to B, signed with her secret key SK_A. This is a guarantee that only A could have produced such an encryption. This means that A cannot deny to have sent such a message to B (note that B is also contained into the signature). Then B sends back to A message M signed with its own secret key, in order to confirm the reception of M. After that, also B will not be able to deny that he has received M from A. We can imagine that M represents some form of electronic payment: A wants a proof that B has received from her the money (a receipt) and B wants a proof that A had indeed sent the payment M to him (for example, in case M is not a valid payment). These are non-repudiation properties and are guaranteed by the signatures (assuming that neither A nor B publicizes her/his secret key). For example, if A collects a certain evidence that B has sent a message (in our protocol the signature $\{M, A\}_{SK_B}$ represents such an evidence), then B should have indeed sent such a message. This is somehow similar to the entity authentication property we have discussed above. The main difference is that B can be malicious and may try to send a faked evidence.

Consider now the fairness property. Indeed, in the simple protocol we have presented, B has an evident advantage over A. He can just refuse to send the last message and A will never be able to prove that she has indeed sent the money to B. The exchange is clearly not fair. We can try to define non-repudiation with fairness as follows:

> "P guarantees non-repudiation with fairness to A on a message M if, whatever malicious B is considered, if B gets evidence that A has originated M than also A will eventually obtain the evidence that B has received M".

We observe two important points. First of all, in this definition one of the parties (B) takes the role of the hostile environment. This is intuitively correct, since we want to see if B is able to cheat A by keeping the payment without releasing the receipt. The second point is the most important: this property cannot be expressed as a safety property (i.e., nothing bad happens). As a matter of fact we are requiring that something good should happen if B gets his evidence, i.e., that also A should soon or later get her evidence. This leads us to relax our general scheme as follows:

> "P guarantees non-repudiation with fairness to A on a message M if, whatever hostile environment is considered (with a certain initial knowledge ϕ_I), then P *satisfies* the specification $\alpha(P)$",

where B is the enemy (ϕ_I is then B's initial knowledge), the relation *satisfies* is a deadlock-sensitive process preorder (e.g., the testing preorder [7]) and $\alpha(P)$ is the process where every time B gets his evidence then also A gets her own evidence. The choice of a deadlock-sensitive preorder is justified by the fact that we want to check that no deadlock is possible after B evidence and this is what we want to require also in protocol P. In the example above, if B decides not to send the second message we clearly obtain such a deadlock and the property is not satisfied.

3 A General Scheme for Security Properties

We have seen that several different properties (message authenticity, entity authentication, secrecy) can be (informally) written in a similar style which sounds like:

> "P guarantees a security property S if, whatever hostile environment is considered, P never shows some particular *bad behaviour*".

In general, this set of bad behaviours depends on the particular property and sometimes may also depend on the protocol P. For example, for message authenticity we need the parameter m of P in order to define what is a bad behaviour. It is sometimes easier to choose a complementary approach and describe which are the good behaviors (they just correspond to all the behaviour that are not bad). So, if we denote by $\alpha_S(P)$ the set of all possible good behaviour of P with respect to the security property S, then our general scheme becomes the following:

> "P guarantees a security property S if, whatever hostile environment is considered, P always shows behaviours in $\alpha_S(P)$".

Indeed, cryptographic protocols typically rely on some secret values (keys or random numbers used for challenge-response). Moreover, if we want to analyze secrecy we certainly have to face the presence of some initially secret message. We can thus slightly refine our scheme as follows:

> "P guarantees a security property S if, whatever hostile environment is considered with a certain initial knowledge ϕ_I, then P always shows behaviours in $\alpha_S(P)$".

Moreover, as discussed for fair non repudiation, it may be useful also to parameterize the previous notion w.r.t. the notion of behaviour, by considering *satisfaction* relations among processes, hence obtaining:

> "P guarantees a security property S if, whatever hostile environment is considered with a certain initial knowledge ϕ_I, then P *satisfies* the specification $\alpha_S(P)$",

The (informal) considerations above are at the base of the proposal, originally reported in [15], of a uniform formal framework where security properties can be defined. The proposed schema, called General NDC ($GNDC$ for short), is as follows: [4]

$$P \text{ is } GNDC^{\alpha}_{\approx} \text{ iff } \forall X \in \mathcal{E}^{\phi_I}_{C} : (P \parallel X) \setminus C \approx \alpha(P)$$

where $\approx$ is a behavioural preorder and α is a function from processes to processes. Now, we can just define a specific property by suitably instantiating the function $\alpha(P)$ and the preorder $\approx$. We reconsider the properties presented so far, showing informally their corresponding $\alpha(P)$ functions and $\approx$ relations:

- *Non-interference*: $\alpha_{NI}(P(M)) = P(M) \setminus C$. We obtain the exact definition of NDC if we use trace equivalence as $\approx$.
- *Message authenticity*: $\alpha_{MA}(P(M))$ is the process where $received(M)$ is the only event $received$ which may occur. For a formal characterisation of this property in the $GNDC$ scheme for a large class of protocols, please see [13], where the $\approx$ relation is (a suitable) may testing or trace equivalence.
- *Entity authentication*: $\alpha_{EA}(P)$ is the process where $commit(A, B)$ is always preceded by $run(B, A)$. Please refer to [15] for other authentication properties based on the *correspondence* idea, such as the ones in the hierarchy of [21] or message authentication as proposed in [24]. The relation $\approx$ is in general trace inclusion.
- *Secrecy of m*: $\alpha_{Sec}(P(m))$ is the set of processes where the event $learnt(m)$ can never occur (for more details, see [14]). The relation $\approx$ is in general trace inclusion.
- *Non repudiation*: $\alpha_{nr}(P(M))$ is the process where whenever an evidence of a message M is obtained then that message has been effectively sent (see [15] for a deeper discussion). The relation $\approx$ is in general trace inclusion.
- *fairness*: $\alpha_{fair}(P(M))$ is the process such that if the event $B_ev_A_or_M$ (signaling that B has evidence that A originated M) then the event $A_ev_B_rec_M$ (signaling that A has evidence that B received M) will eventually happen (see [15] for a deeper discussion). The relation $\approx$ is in general a failure or testing preorder (see [7,19]).

3.1 Security Attacks as Interferences

In the previous section we have shown how several security properties can be seen as instances of the following general1 scheme:

> "P guarantees a security property S if, whatever hostile environment is considered with a certain initial knowledge ϕ_I, then P *satisfies* the specification $\alpha_S(P)$",

where the relation *satisfies* and the function $\alpha_S(P)$ are property dependent parameters.

[4] Indeed $GNDC$ depends on the set ϕ_I, but we will omit it for the sake of readability.

It would be useful to find the most restrictive $\alpha_S(P)$ as it would induce the strongest property, up to the chosen notion of behaviour (i.e., the chosen relation *satisfies*). The idea is to use an $\alpha_S(P)$ which returns an *encapsulation* of protocol P, i.e., a version of P which is completely isolated from the environment. Intuitively, this secure *encapsulation* of P should correspond to the execution of P in a perfectly secure network where only the honest parties are present. In our process algebra setting, this corresponds to the *restriction* of all public channels where protocol messages are sent. This makes it impossible for an intruder to *interfere* on the protocol execution. Let us briefly explain this important point. Consider a preorder between processes $\leq$. Next, consider the induced equivalence $\approx$ as $\leq \cap \leq^{-1}$. Also suppose that for every process P we have

$$(P \parallel \underline{0}) \setminus C \approx P \setminus C$$

where $\underline{0}$ is the process that does nothing. This means that the process restricted on C is equivalent to the protocol in composition with the intruder that does nothing. The previous property holds for every security property we have studied. Please also note that, by definition, $\underline{0} \in \mathcal{E}_C^{\phi_I}$ for every ϕ_I. So it is very natural to consider α functions and processes P such that:

$$P \setminus C \leq \alpha(P)$$

This simply means that the protocol P is correct (as it satisfies its specification $\alpha(P)$) at least when it is not under attack. This condition can be somehow seen as a reasonable criterion for any *good* protocol: it must be correct at least when it is not under attack! Under this observation, it is clear that $P \in NDC$ implies $P \in GNDC^{\alpha}_{\leq}$.

If NDC holds then we can say that the hostile environment has no effect at all on P, since it still behaves as if it were isolated. It is important to note that what we observe of P behaviour are exactly the events we have discussed in the previous section. It is easy to see that, if we specify all the events corresponding to a certain set of properties, then NDC will imply all of them. [5]

When P does not guarantee NDC we say that an *interference* is possible, i.e., a behaviour that is caused by the hostile environment. It comes out that an attack to a security properties is revealed by NDC as an interference of the enemy on the protocol. This allows us to use NDC in order to detect different attacks all at once. In the next section we illustrate this issue with an example.

4 An Example

In this section we consider a larger protocol and we show how the analysis of (some of) the security properties presented above, can be carried out in a uniform way. The protocol is the Woo-Lam public key one, which has been proposed for

[5] Note that, the notion of *satisfies* must be chosen as the stronger one used by the security properties considered.

mutual entity authentication and key-exchange. The protocol, as reported in [27], is the following sequence of 7 messages:

Message 1	$A \to KDC$:	A, B
Message 2	$KDC \to A$:	$\{PK_B\}_{SK_{KDC}}$
Message 3	$A \to B$:	$\{A, N_A\}_{PK_B}$
Message 4	$B \to KDC$:	$A, B, \{N_A\}_{PK_{KDC}}$
Message 5	$KDC \to B$:	$\{PK_A\}_{SK_{KDC}}, \{\{N_A, K, A, B\}_{SK_{KDC}}\}_{PK_B}$
Message 6	$B \to A$:	$\{\{N_A, K, A, B\}_{SK_{KDC}}, N_B\}_{PK_A}$
Message 7	$A \to B$:	$\{N_B\}_K$

Alice sends to KDC a request of connection with Bob. Then KDC replies with a certified copy of B's public key. This copy is indeed signed with KDC's secret key, i.e., only KDC may have generated it. Alice checks the signature and sends to Bob a challenge N_A encrypted with Bob's public key. Bob forwards N_A to KDC encrypting it with KDC's public key and adding both his own identifier and the one of Alice. KDC is now ready to generate a certificate containing the challenge N_A, the fresh session key K and the two identifiers A and B. KDC sends this to Bob (encrypted with Bob's public key) together with a signed copy of Alice's public key. Bob checks the signature and forwards to Alice the certificate received from KDC together with a challenge N_B, all encrypted with Alice's public key. Finally, Alice sends back to Bob the challenge N_B encrypted with the new session key K.

This protocol looks quite complex. As a matter of fact it mixes the requests for public keys with entity authentication and key-exchange. In particular the first two messages and the first part of message 5 are for public keys distribution. The challenges N_A and N_B are used to guarantee mutual entity authentication. Finally, the certificate provides (authenticated) key-distribution. Indeed, this protocol contains two errors with respect to the correct version [29] that cause a number of attacks. We now analyze the protocol by applying the ideas developed in the previous section. First of all we point out in detail which are the various security properties that the protocol should guarantee:

- public key PK_A and PK_B should be authentic from KDC (this is why they are signed);
- it is also important that the session key K is authentic, i.e., no enemy should be able to force A and B using a faked session key;
- of course, the session key K should also remain secret;
- also the challenges should remain secret (since they are always sent encrypted) but this is not crucial since they are used only for guaranteeing entity authentication; in protocols where the nonces are also used for generating a new session key secrecy requirement becomes crucial;
- finally, the protocol should guarantee mutual entity authentication between A and B.

We now show all the events that are used to model such properties as suitable annotation to the protocol, where the secrecy event $learnt(K)$ is not reported,

Table 1. An attack to secrecy and message authenticity

1	$A \to KDC$:	A, B	$run(A, B)$
2	$KDC \to A$:	$\{PK_B\}_{SK_{KDC}}$	$received(PK_B)$
3	$A \to B$:	$\{A, N_A\}_{PK_B}$	$run(B, A)$
4	$B \to KDC$:	$A, B, \{N_A\}_{PK_{KDC}}$	
5	$KDC \to \mathbf{E}(B)$:	$\{PK_A\}_{SK_{KDC}}$, $\{\{N_A, K, A, B\}_{SK_{KDC}}\}_{PK_B}$	
5′	$\mathbf{E}(KDC) \to B$:	$\{\mathbf{PK_E}\}_{\mathbf{SK}_{KDC}}$, $\{\{N_A, K, A, B\}_{SK_{KDC}}\}_{PK_B}$	$received(\mathbf{PK_E})$
6	$B \to \mathbf{E}(A)$:	$\{\{N_A, K, A, B\}_{SK_{KDC}}, N_B\}_{\mathbf{PK_E}}$	**learnt(K)**
6′	$\mathbf{E}(B) \to A$:	$\{\{N_A, K, A, B\}_{SK_{KDC}}, N_B\}_{\mathbf{PK_A}}$	$received(K)$ $commit(A, B)$
7	$A \to B$:	$\{N_B\}_K$	$commit(B, A)$

as it is performed by the omitted enemy. This allows us check all the properties in just one step following our NDC approach.

1	$A \to KDC$:	A, B	$run(A, B)$
2	$KDC \to A$:	$\{PK_B\}_{SK_{KDC}}$	$received(PK_B)$
3	$A \to B$:	$\{A, N_A\}_{PK_B}$	$run(B, A)$
4	$B \to KDC$:	$A, B, \{N_A\}_{PK_{KDC}}$	
5	$KDC \to B$:	$\{PK_A\}_{SK_{KDC}}$, $\{\{N_A, K, A, B\}_{SK_{KDC}}\}_{PK_B}$	$received(PK_A)$
6	$B \to A$:	$\{\{N_A, K, A, B\}_{SK_{KDC}}, N_B\}_{PK_A}$	$received(K)$ $commit(A, B)$
7	$A \to B$:	$\{N_B\}_K$	$commit(B, A)$

The security properties that this protocol should guarantee have in common that trace inclusion is their suitable $\approx$ relation. Hence, an interference is a trace that is composed only by the security events and that is possible when the enemy is active, but not when the protocol runs in isolation. Consider now the attack sequence in Table 1, where the interference trace is reported in the right column. In this attack E exploits (one of) the mistakes in the specification of the protocol as reported in [27]. In particular, it is necessary to include in the certificate of the public key also the identifier of the corresponding owner. Hence, in messages 2 and 5 the certificate should be $\{PK_B, B\}_{SK_{KDC}}$ and $\{PK_A, A\}_{SK_{KDC}}$, respectively. The enemy can thus substitute in message 5 the signed public key of A with its own signed key. Note that the enemy can obtain a signed copy of its own key by just running honestly the protocol with another user. After that, the sixth message will be encrypted by Bob with the public key of the enemy thus allowing the interception of the certificate and, consequently, of the session key K. By observing the events we can identify two attacks:

1. the protocol does not guarantee the authenticity of Alice public key in message 5; we observe this through event $received(PK_E)$; as we stated above, this is caused by the absence of Alice identifier inside the signature;

Table 2. An attack to secrecy, authenticity and entity authentication

1	$\mathbf{E}(A) \rightarrow KDC$:	A, B	
2	$KDC \rightarrow \mathbf{E}(A)$:	$\{PK_B\}_{SK_{KDC}}$	
3	$\mathbf{E}(A) \rightarrow B$:	$\{A, N_A\}_{PK_B}$	$run(B, A)$
4	$B \rightarrow KDC$:	$A, B, \{N_A\}_{PK_{KDC}}$	
5	$KDC \rightarrow \mathbf{E}(B)$:	$\{PK_A\}_{SK_{KDC}}$, $\{\{N_A, K, A, B\}_{SK_{KDC}}\}_{PK_B}$	
5′	$\mathbf{E}(KDC) \rightarrow B$:	$\{\mathbf{PK_E}\}_{\mathbf{SK}_{KDC}}$, $\{\{N_A, K, A, B\}_{SK_{KDC}}\}_{PK_B}$	$received(\mathbf{PK_E})$
6	$B \rightarrow \mathbf{E}(A)$:	$\{\{N_A, K, A, B\}_{SK_{KDC}}, N_B\}_{\mathbf{PK_E}}$	**learnt(K)**
7	$\mathbf{E}(A) \rightarrow B$:	$\{N_B\}_K$	$commit(B, A)$

2. the protocol does not guarantee the secrecy of K; this is revealed by event $learnt(K)$.

Note also that we do not observe any entity authentication attack. Alice and Bob are convinced to communicate one another at the end of the protocol. Thus, the secrecy attack over K becomes even more dangerous, as the enemy can easily eavesdrop every future communication encrypted with K between Alice and Bob and they have no way of discovering this.

Indeed it is easy to show an attack similar to the previous one in order to obtain also an entity authentication failure, e.g. see Table 2.

The idea is that the role plaied by A in the previous attack could be fully simulated by the enemy as done here. Since we have $commit(B, A)$ with no $run(A, B)$ the entity authentication attack is indeed revealed. Hence, the protocol does not guarantee the authentication of A with respect to B, i.e., the enemy is indeed able to impersonate A. Note that the attack on message authenticity and secrecy are still valid.

Other complex attacks are possible, involving two or three parallel sessions of the protocol. All of them are based on the fact that the certificate does not include the corresponding identifier. As we already stated, this is an error in the version reported in [27]. In the original protocol [28] the certificates are correct. However in such a version there is another error, causing different failures, that has been corrected two months later by the same authors in [29].

5 Conclusion

Our non interference-based approach to the analysis of protocols has been mechanized [8,9], resulting in a tool that checks NDC on finite state representations of the honest participants and the (most general) enemy. With the help of this tool, we have been able to show failures upon two unflawed (to the best of our knowledge) protocols: Woo & Lam public key one-way authentication protocol and ISO public key two-pass parallel authentication protocol; and new failures upon three flawed protocols: Encrypted Key Exchange, Station to Station, Woo

& Lam symmetric key one-way authentication protocol (the last one reported in [8,9]). Many other protocols have been analyzed, most of those reported in the cryptographic protocol library [6], and we have been able to capture the attacks reported there. We are now improving the efficiency of our tool in order to be able to analyze larger, commercial protocols for e-commerce, such as SET [22].

Future extensions of the approach include the modeling of cryptographic protocols with more concrete information, e.g., time and probability, that can be helpful in order to discover time/probability dependent attacks that cannot be revealed in a purely nondeterministic setting. Some initial work in this direction is [12], where the NDC idea has been extended in the context of a discrete time process algebra, and applied to prevent timing covert channels in multilevel computer systems.

Our NDC-based approach has been developed for a CCS-like calculus that is powerful enough to model most cryptographic protocols. However, recursive protocols as well as protocols for mobile systems (where channel names are passed as values in a communication) cannot be easily modeled. Hence, we are planning to study the extension of our approach to richer calculi, such as the spi-calculus [2].

References

1. M. Abadi. Security protocols and specifications. In *Proc. Foundations of Software Science and Computation Structures*, volume 1578 of *LNCS*, pages 1–13, 1999.
2. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148(1):1–70, 1999.
3. M. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, January 1996.
4. D. E. Bell and L. J. La Padula. Secure computer systems: Unified exposition and multics interpretation. *ESD-TR-75-306, MITRE MTR-2997*, March 1976.
5. C. Bodei, P. Degano, R. Focardi, and C. Priami. Authentication via localized names. In *Proceedings of CSFW'99*, pages 98–110. IEEE press, 1999.
6. J. Clark and J. Jacob. "A Survay of Authentication Protocols Literature: version 1.0". November 1997.
   http://www.cs.york.ac.uk/∼jac/papers/drareview.ps.gz.
7. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.
8. A. Durante, R. Focardi, and R. Gorrieri. A compiler for analysing cryptographic protocols using non-interference. Submitted for publication.
9. A. Durante, R. Focardi, and R. Gorrieri. CVS: A compiler for the analysis of cryptographic protocols. In *Proceedings of CSFW'99*, pages 203–212. IEEE press, 1999.
10. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1994/1995.
11. R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9), September 1997.
12. R. Focardi, R. Gorrieri, and F. Martinelli. Information flow analysis in a discrete-time process algebra. In *Proceedings of CSFW'00*, 2000. IEEE Press. To appear.

13. R. Focardi, R. Gorrieri, and F. Martinelli. Message authentication through non-interference. In *Proc. of 8th International Conference in Algebraic Methodology and Software Technology* (AMAST), 2000. To appear.
14. R. Focardi, R. Gorrieri, and F. Martinelli. Secrecy in security protocols as non-interference. In *Workshop on secure architectures and information flow*, volume 32 of *ENTCS*, 2000.
15. R. Focardi and F. Martinelli. A uniform approach for the definition of security properties. In *Proceedings of World Congress on Formal Methods (FM'99)*, pages 794–813. Springer, LNCS 1708, 1999.
16. J. A. Goguen and J. Meseguer. Security policy and security models. In *Proc. of the 1982 Symposium on Security and Privacy*, pages 11–20. IEEE Press, 1982.
17. D. Gollman. What do we mean by entity authentication? In *Proceedings of Symposium in Research in Security and Privacy*, pages 46–54. IEEE Press, 1996.
18. D. Gollman. On the verification of cryptographic protocols - a tale of two committees. In *Workshop on secure architectures and information flow*, volume 32 of *ENTCS*, 2000.
19. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.
20. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Proceedings of TACAS'96*, pages 146–166. LNCS 1055, 1996.
21. G. Lowe. A hierarchy of authentication specification. In *Proceedings of the 10th Computer Security Foundation Workshop*, pages 31–43. IEEE press, 1997.
22. MasterCard. Secure electronic payment protocol, November 1995. Draft Version 1.2.
23. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
24. S. Schneider. Verifying authentication protocols with csp. In *Proceedings of the 10th Computer Security Foundation Workshop*, pages 3–17. IEEE press, 1997.
25. S. Schneider. Formal analysis of a non-repudiation protocol. In *Proceedings of CSFW'98*, pages 54–65. IEEE Press, 1998.
26. S. Schneider. Verifying authentication protocols in CSP. *IEEE Transactions on Software Engineering*, 24(9), September 1998.
27. F.B. Schneier. *Applied Cryptography.* Wiley, 1996.
28. T. Y. C. Woo and S. S. Lam. Authentication for distributed systems. *IEEE Computer*, 25(1):39–51, 1992.
29. T. Y. C. Woo and S. S. Lam. Authentication revised. *IEEE Computer*, 25(3):10, 1992.
30. T. Y. C. Woo and S. S. Lam. A semantic model for authentication protocols. In *Proceedings of the 1993 IEEE Computer Society Symposium on Security and Privacy (SSP '93)*, pages 178–195. IEEE Press, May 1993.
31. J. Zhou and D. Gollman. A fair non-repudiation protocol. In *Proc. of Symposium in Research in Security and Privacy*, pages 55–61. IEEE Press, 1996.

Average Bit-Complexity of Euclidean Algorithms

Ali Akhavi and Brigitte Vallée

GREYC, Université de Caen, F-14032 Caen (France)
Ali.Akhavi@info.unicaen.fr, Brigitte.Vallee@info.unicaen.fr

Abstract. We obtain new results regarding the precise average bit-complexity of five algorithms of a broad Euclidean type. We develop a general framework for analysis of algorithms, where the average-case complexity of an algorithm is seen to be related to the analytic behaviour in the complex plane of the set of elementary transformations determined by the algorithms. The methods rely on properties of transfer operators suitably adapted from dynamical systems theory and provide a unifying framework for the analysis of an entire class of gcd-like algorithms.

1 Introduction

Motivations. Euclid's algorithm was analysed first in the worst case in 1733 by de Lagny, then in the average-case around 1969 independently by Heilbronn [12] and Dixon [6], and finally in distribution by Hensley [13] who proved in 1994 that the Euclidean algorithm has Gaussian behaviour. The first methods used range from combinatorial (de Lagny, Heilbronn) to probabilistic (Dixon). In parallel, studies by Lévy, Khinchin, Kuzmin and Wirsing had established the metric theory of continued fractions by means of a specific density transformer. The more recent works rely for a good deal on *transfer operators*, a far-reaching generalization of density transformers, originally introduced by Ruelle [18,19] in connection with the thermodynamic formalism and dynamical systems theory [1]. Examples are Mayer's studies on the continued fraction transformation [15], Hensley's work [13] and several papers of Vallée [21,22,23,24].

All the previous analyses deal with the number of arithmetical operations performed during the execution of the algorithm. In this paper, we provide new analyses that characterize the precise average bit-complexity of a class of Euclidean algorithms.

We consider here five algorithms that are all classical variations of the Euclidean algorithm and are called *Classical* ($\mathcal{G}$), *By-Excess* ($\mathcal{L}$), *Centered* ($\mathcal{K}$), *Subtractive* ($\mathcal{T}$) and *Binary* ($\mathcal{B}$). The complexity of these algorithms (in terms of the number of arithmetical operations to be performed) is now well–known: The two most common algorithms ($\mathcal{G}$) and ($\mathcal{K}$) have been analysed by Heilbronn [12], Dixon [6] and Rieger [17]. The Subtractive algorithm ($\mathcal{T}$) was studied by Yao and Knuth [26], and Vardi [25] analysed the By-Excess Algorithm ($\mathcal{L}$) by comparing it to the Subtractive Algorithm. Brent [3] and Vallée [23] have analysed the Binary algorithm ($\mathcal{B}$).

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 373–387, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Methods. Our approach is a refinement of methods that have been already used for instance in [4,9,23,24]: it consists in viewing an algorithm of the broad gcd type as a dynamical system, where each iterative step is a linear fractional transformation (LFT) of the form $z \to (az+b)/(cz+d)$. A specific set of transformations is then associated with each algorithm. It already appears from previous treatments that the computational complexity of an algorithm is in fact dictated by the collective dynamics of its associated set of transformations.

A previous work [24] describes a classification of gcd-like algorithms in terms of the average number of arithmetical operations: some of them are fast, that is, of logarithmic complexity $\Theta(\log N)$, while others are slow, that is, of the log-squared type $\Theta(\log^2 N)$. It was established there that strong contraction properties of the elementary transformations that build up a gcd-like algorithm entail logarithmic cost, while the presence of an indifferent fixed-point leads to log-squared behaviour.

It is not a priori clear whether the previous classification between fast algorithms and slow algorithms gives access to the average bit–complexity. The reason is that, even if fast algorithms perform fewer iterations, each iteration is often more complex than in the case of slow algorithms. In this paper, we prove that, in terms of the average-bit-complexity, fast algorithms are of log-squared type $\Theta(\log^2 N)$ while slow ones are of log-cubed type $\Theta(\log^3 N)$. Our approach also precisely determines the constants that intervene in the expected costs. They are closely related to the main characteristics of the associated dynamical system (entropy, invariant measure, ...). These constants are computable numbers though they are not always related to classical constants of analysis. Our method can also open access (it will be shown in the full paper) to characteristics of the distribution of bit-complexity costs, including information on moments: the fast algorithms appear to have concentration of distribution —the cost converges in probability to its mean— while the slow ones exhibit an extremely large dispersion of costs.

Technically, this paper relies on a description of relevant parameters by means of generating functions, a common tool by now in the average-case analysis of algorithms [7,8]. As is usual in number theory contexts, the generating functions are Dirichlet series. They are first proved to be algebraically related to specific operators that encapsulate all the important informations relative to the "dynamics" of the algorithm. Their analytical properties depend on spectral properties of the operators, most notably the existence of a "spectral gap" that separates the dominant eigenvalue from the remainder of the spectrum. This determines the singularities of the Dirichlet series of costs. The asymptotic extraction of coefficients is then achieved by means of Tauberian theorems, one of the many ways to derive the prime number theorem. Average bit–complexity estimates finally result. The main thread of the paper is thus summarized by the chain:

Euclidean algorithm $\rightsquigarrow$ Associated transformations $\rightsquigarrow$ Transfer operator $\rightsquigarrow$ Dirichlet series of costs $\rightsquigarrow$ Tauberian inversion $\rightsquigarrow$ Average-case complexity.

This chain then leads to effective and simple criteria for distinguishing slow algorithms from fast ones and for establishing concentration of distribution, etc.

Results and plan of the paper. Section 3 is the central technical section of the paper. There, we develop the line of attack outlined earlier and introduce successively Dirichlet generating functions, transfer operators of the Ruelle type, and the basic elements of Tauberian theory that are adequate for our purposes. The main results of this section are summarized in Theorem 1 that describes the singularities of generating functions of bit–costs and implies a general criterion for log-squared versus log-cubed behaviour.
In Section 4, we return to our five favorite algorithms. The corresponding analyses are summarized in Theorems 2 and 3 where we state our main results that fall as natural consequences of the present framework. It results from the analysis (Theorem 2) that the algorithms of the Fast Class —the Classical Algorithm ($\mathcal{G}$), the Centered Algorithm ($\mathcal{K}$), and the Binary algorithm ($\mathcal{B}$)— when applied to random integers less than N, have average bit-complexity of the form $B_N(\mathcal{H}) \sim A(\mathcal{H}) \log_2^2 N$, with $\mathcal{H} \in \{\mathcal{G}, \mathcal{K}, \mathcal{B}\}$. Each of the three constants $A(\mathcal{G}), A(\mathcal{K}), A(\mathcal{B})$ is a product of two constants: the first one is a constant à la Lévy and is effectively characterized as the inverse of the entropy of the associated dynamical system, while the second one is a constant à la Khinchin. The constants related to the two classical algorithms are explicit and easily obtained,

$$A(\mathcal{G}) = \frac{6 \log^2 2}{\pi^2} \left[2 + \frac{1}{\log 2} \log \prod_{k=0}^{\infty} (1 + \frac{1}{2^k})\right],$$

$$A(\mathcal{K}) = \frac{6 \log \phi \log 2}{\pi^2} \left[3 + \frac{\log 2}{\log \phi} + \frac{1}{\log \phi} \log \prod_{k=3}^{\infty} \frac{(2^k - 1)\phi^2 + 2\phi}{(2^k - 1)\phi^2 - 2}\right].$$

The constant relative to the Binary Algorithm is expressed in terms of the invariant measure $\psi_2(t)dt$ and its distribution function $F_2(t)$ (that are not explicit in this case) as

$$A(\mathcal{B}) = \frac{2 \log 2}{\pi^2 \, \psi_2(1)} \left[1 + \sum_{m \text{ odd} \geq 1} \frac{1}{2^{\ell(m)}} F_2(\frac{1}{m})\right],$$

where $\ell(m)$ denotes the binary length of integer m. Exact computations (for the first two algorithms) or various batches of a few thousands simulations on numbers of order of 10^{100} (for the Binary Algorithm) suggest numerical values for the three constants $A(\mathcal{G}) \simeq 1.24237$ $A(\mathcal{K}) \simeq 1.12655$ $A(\mathcal{B}) \simeq 0.7$. These values prove the efficiency of fast Euclidean Algorithms, compared to naive multiplication whose average bit-complexity is $\log_2^2 N$ on integers less than N.
Theorem 3 proves that the algorithms of the Slow Class —the By-Excess Algorithm ($\mathcal{L}$) and the Subtractive Algorithm ($\mathcal{T}$)— have average bit-complexity of the log–cubed type, $B_N(\mathcal{H}) \sim A(\mathcal{H}) \log_2^3 N$ with $\mathcal{H} \in \{\mathcal{L}, \mathcal{T}\}$, and

$$A(\mathcal{T}) = \frac{2 \log^2 2}{\pi^2}, \quad A(\mathcal{L}) = \frac{\log^2 2}{\pi^2}.$$

2 Five Variations of the Euclidean Algorithm

We present the five algorithms to be analysed; the first three use divisions, while the last two use only simpler operations, as subtractions and/or right shifts.

2.1. Euclidean Algorithms with divisions. There are two divisions between u and v ($v > u$), that produce a positive remainder r such that $0 \leq r < u$: the classical division (by-default) of the form $v = mu + r$, and the division by-excess, of the form $v = mu - r$. The centered division between u and v ($v > u$), of the form $v = mu + \varepsilon r$, with $\varepsilon = \pm 1$ produces a positive remainder r such that $0 \leq r < u/2$. There are three Euclidean algorithms associated with each type of division, respectively called the Classical Algorithm ($\mathcal{G}$), the By-Excess Algorithm ($\mathcal{L}$), and the Centered Algorithm ($\mathcal{K}$).
We denote by $\ell(x)$ the number of bits of the positive integer x. Then, the bit–cost of a division step, of the form $v = mu + \varepsilon r$ is equal to $\ell(u) \times \ell(m)$. It is followed by exchanges which involve numbers u and r, so that the total cost of a step is $\ell(u) \times \ell(m) + \ell(u) + \ell(r)$. In the case of the centered division, there is possibly a supplementary subtraction (in the case when $\varepsilon = -1$) in order to obtain a remainder in the interval $[0, u/2]$.

2.2. Euclidean Algorithms without divisions. On the other hand, there are two algorithms where no divisions are performed, the Subtractive Algorithm ($\mathcal{T}$) and the Binary Algorithm ($\mathcal{B}$).
The Subtractive Algorithm uses only subtractions, since it replaces the classical division $v = mu + r$ by a sequence of m subtractions of the form $v := v - u$. The cost of a subtractive step $v = u + (v - u)$ is equal to $\ell(v)$. Then the bit–cost of a sequence of a sequence of subtractions equals $\ell(v) \times m$. It is followed by an exchange, so that the total cost of a sequence of m subtractions $\ell(v) \times (m + 2)$ for the Subtractive algorithm.
The Binary Algorithm uses only subtractions and right shifts, since it performs operations of the form $v := (v - u)/2^b$, where b is the dyadic valuation of $v - u$, denoted by $b := \mathrm{Val}_2\,(v - u)$, and defined as the largest exponent b such that 2^b divides $v - u$. This algorithm has two nested loops and each external loop corresponds to an exchange. Between two exchanges, there is a sequence of (internal) iterations that constitutes one external step.

Binary Euclidean Algorithm (u, v)
While $u \neq v$ **do**
  While $u < v$ **do**
    $b := \mathrm{Val}_2\,(v - u)$; $v := (v - u)/2^b$;
  Exchange u and v;
Output: u (or v).

Each internal step consists in subtractions and shifts and a sequence of internal steps can be written as

$$v = u + 2^{b_1} v_1, \qquad v_1 = u + 2^{b_2} v_2, \qquad \ldots \qquad v_{\ell-1} = u + 2^{b_\ell} v_\ell, \tag{1}$$

Here v_ℓ is strictly less than u, and plays the rôle of a remainder r, so that the result of the sequence (1) is a decomposition of the form $v = mu + 2^s r$, with m odd, $m < 2^s$ and $r < u$ and constitutes an external step. The number of internal steps in (1) equals $b(m)$, where $b(x)$ denotes the number of ones in the binary expansion of x.
The cost of a shift $v := v/2^b$ is equal to $\ell(v)$. Then the bit–cost of a sequence of internal steps whose result is a decomposition $v = mu + dr$ equals $\ell(v) \times b(m)$. It is followed by an exchange, so that the total cost of an external step is $\ell(v) \times (b(m) + 2)$ for the Binary Algorithm.

Alg., Type	*Division*	*Set $\mathcal{H}$ of LFT's*	*Final Set $\mathcal{F}$*	*Cost of the LFT.*
($\mathcal{G}$) (1, all, 0)	$v = mu + r$ $0 \le r < u$	$\{\frac{1}{m+x}, m \ge 1\}$	$m \ge 2$	$\ell(m) + 2$
($\mathcal{L}$) (1, all, 1)	$v = mu - r$ $0 \le r < u$	$\{\frac{1}{m-x}, m \ge 2\}$	$m \ge 3$	$\ell(m) + 2$
($\mathcal{K}$) ($\frac{1}{2}$, all, 0)	$v = mu + \varepsilon r$ $m \ge 2, \varepsilon = \pm 1,$ $(m, \varepsilon) \ne (2, -1)$ $0 \le r < \frac{u}{2}$	$\{\frac{1}{m+\varepsilon x},$ $m \ge 2, \varepsilon = \pm 1\}$	$\varepsilon = +1$	$\ell(m) + 2 + \frac{1-\varepsilon}{2}$
($\mathcal{T}$) (1, all, 0)	$v = mu + r$ $0 \le r < u$	$\{\frac{1}{m+x}, m \ge 1\}$	$m \ge 2$	$m + 2$
($\mathcal{B}$) (1, odd, 1)	$v = mu + 2^s r,$ m odd, $m < 2^r, r < u$	$\{\frac{1}{m+2^s x},$ m odd, $m < 2^s\}$	$\mathcal{F} = \mathcal{H}$	$b(m) + 2$

Fig. 1. The five Euclidean algorithms.

2.3. The sets of linear fractional transformations. When given an input (u_1, u_0), $(u_1 \le u_0)$, each of the five algorithms performs k (external) steps of the form
$u_0 = m_1 u_1 + d_1 u_2, \qquad u_1 = m_2 u_2 + d_2 u_3, \qquad \ldots u_{k-1} = m_k u_k + d_k u_{k+1},$
and decomposes the rational $x := (u_1/u_0)$ as $(u_1/u_0) = h_1 \circ h_2 \circ \ldots \circ h_k(a)$, where the h_i's are linear fractional transformations (LFT) of the form $h_i = h_{[m_i, d_i]}$ with $h_{[m,d]}(x) = 1/(m + dx)$ and $a := (u_{k+1}/u_k)$ is the last value of the rational.
The precise form of the possible LFT's depends on the algorithm; there may exist a special set $\mathcal{F}$ of LFT's in the final step: for instance, in the Classical Algorithm ($\mathcal{G}$), the last quotient $m = 1$ is forbidden. However, all the other steps use the same set of LFT's, that we call the generic set.
The value a equals 1 for the By-Excess Algorithm ($\mathcal{L}$) and the Binary Algorithm ($\mathcal{B}$), and equals 0 for the other three algorithms. The rational inputs of each algorithm belong to the basic interval $\mathcal{I} = [0, \rho]$ with $\rho = 1$ or $\rho = 1/2$: For the centered algorithm ($\mathcal{K}$), one has $\rho = 1/2$ and otherwise $\rho = 1$. For the first four algorithms, the valid inputs are all the rationals of $\mathcal{I}$, while the valid inputs of the last algorithm are only the odd rationals of $\mathcal{I}$. The variable "valid" has two

possible values {all, odd}, and finally, the type of the algorithm is defined as the triple (ρ, valid, a).

In all the cases, when performing k (external) steps on the input (u_1, u_0), the bit-cost $C(u_1, u_0)$ of the algorithm is a sum of k terms, the i-th term representing the cost of the i-th (external) step and being a product of two factors; the first factor involves the binary length $\ell(u_j)$ of integer u_j (with j possibly equal to $i-1, i$ or $i+1$ according to the precise algorithm), while the second one involves a cost relative to the i-th LFT to be performed, of the form $c(h_i)$. In the sequel, we can replace the length $\ell(u)$ of integer u by its logarithm $\log_2(u)$ in base 2 and always consider $\log_2(u_i)$ as the first factor to be studied. In contrast, we have to work with the exact cost due to the LFT. Finally, the bit-cost $C(u_1, u_0)$ of the algorithm on the input (u_1, u_0) will be always of the form

$$C(u_1, u_0) = \sum_{i=1}^{k} \log_2(u_i) \times c(h_i). \tag{2}$$

The table of Figure 1 describes the precise form of the divisions, the generic set $\mathcal{H}$ of associated LFT's, the final set $\mathcal{F}$ and the cost $c(h)$ of the LFT's that intervene in (2).

3 Generating Functions, Dynamical Operators and Tauberian Theorems

Here, we describe the general tools for analysing bit-complexities of algorithms of the Euclidean type. We first introduce the Dirichlet generating functions relative to the bit-cost of the algorithms, so that the average bit–complexity involves partial sums of coefficients of these Dirichlet series. Tauberian Theorems are a classical tool that transfers analytical behaviour of Dirichlet series near their singularities into asymptotic behaviour of their coefficients. Then, by viewing the algorithm as a dynamical system, we relate generating functions of bit–cost to the Ruelle operator associated with the algorithm, so that we can easily describe the singularities of intervening generating functions.

3.1. Generating functions. The following sets relative to the basic interval $\mathcal{I}$,

$$\widetilde{\Omega} := \{(u,v);\ u,v \text{ valid},\ u/v \in \mathcal{I}\}, \qquad \widetilde{\Omega}_N := \{(u,v) \in \widetilde{\Omega}, v \le N\},$$

$$\Omega := \{(u,v);\ u,v \text{ valid}, \gcd(u,v) = 1, u/v \in \mathcal{I}\}, \qquad \Omega_N := \{(u,v) \in \Omega, v \le N\},$$

are the possible inputs of an algorithm. We denote by $C(u,v)$ the bit-complexity of the algorithm on the input (u,v) as given in (2). We propose to study the average bit-complexity B_N of an algorithm on Ω_N and the average bit-complexity $\widetilde{B}_N$ of an algorithm on $\widetilde{\Omega}_N$, then evaluate the asymptotic behaviour (for $N \to \infty$) of these costs. In fact, it is sufficient to study B_N and this will be shown in the full paper. The Dirichlet generating functions of costs,

$$F(s) := \sum_{(u,v)\in\Omega} \frac{1}{v^s}, \quad G(s) := \sum_{(u,v)\in\Omega} \frac{1}{v^s} C(u,v), \tag{3}$$

are of the form

$$F(s) = \sum_{n\geq 1} \frac{a_n}{n^s}, \qquad G(s) = \sum_{n\geq 1} \frac{c_n}{n^s},$$

where a_n is the number of pairs (u, v) of Ω with fixed $v = n$, and c_n is the cumulative cost on pairs (u, v) of Ω with fixed $v = n$, so that the average cost B_N to be studied is exactly the ratio between partial sums $\sum_{n\leq N} c_n$ of the coefficients of the Dirichlet series $G(s)$ and partial sums $\sum_{n\leq N} a_n$ of the coefficients of the Dirichlet series $F(s)$.

3.2. Tauberian Theorems. The asymptotic evaluation of B_N (for $N \to \infty$) is made possible by the following Tauberian theorem [5], [20] to be applied to the Dirichlet series $F(s)$ and $G(s)$.

Tauberian Theorem. [Delange] *Let $F(s)$ be a Dirichlet series with non negative coefficients such that $F(s)$ converges for $\Re(s) > \sigma > 0$. Assume that*
(i) $F(s)$ is analytic on $\Re(s) = \sigma, s \neq \sigma$, and
(ii) for some $\gamma \geq 0$, one has $F(s) = A(s)(s-\sigma)^{-\gamma-1} + C(s)$, where A, C are analytic at σ, with $A(\sigma) \neq 0$.

Then, as $N \to \infty$,

$$\sum_{n\leq N} a_n = \frac{A(\sigma)}{\sigma\Gamma(\gamma+1)} N^\sigma \log^\gamma N \ \ [1+\varepsilon(N)], \quad \varepsilon(N) \to 0.$$

Theorem applies to the Dirichlet series F, G defined in (3) with $\sigma = 2$. For $F(s)$, it applies with $\gamma = 0$. For $G(s)$, it applies with $\gamma = 2$ or $\gamma = 3$. For the slow algorithms, γ equals 3, and the average bit-complexity will be of order $\log^3 N$. For the fast algorithms, γ equals 2, and the average bit-complexity will be of order $\log^2 N$.
First, the function $F(s)$ is closely linked to the Riemann series of valid numbers $\widehat{\zeta}$, defined as $\widehat{\zeta}(s) := \sum_{v \text{ valid}} v^{-s}$ via the equalities

$$F(s) = \alpha \frac{\widehat{\zeta}(s-1)}{\widehat{\zeta}(s)} \quad \text{with } \alpha = \rho \text{ if valid= all, and } \ \alpha = \frac{\rho}{2} \ \text{ if valid= odd.}$$

Since $\widehat{\zeta}$ is itself easily related to the classical Zeta function, it is then clear that the Tauberian Theorem applies to $F(s)$ with $\sigma = 2$ and $\gamma = 0$. However, it is not clear how to apply directly theTauberian Theorem to $G(s)$. In the following, we obtain expressions for $G(s)$ which involve suitable Ruelle operators and from which the location and the nature of the singularities become apparent.

3.3. Algebraic properties of Ruelle operators. The Ruelle operator $\mathbf{R}_{s,h}$ relative to a LFT h depends on a complex parameter s and is defined as

$$\mathbf{R}_{s,h}[f](x) := \frac{1}{D[h](x)^s} \ f \circ h(x), \tag{4}$$

where $D[h]$ denotes the denominator of the linear fractional transformation (LFT) h, defined for $h(x) = (ax+b)/(cx+d)$ with a, b, c, d coprime integers by $D[h](x) := |cx+d| = |\det h|^{1/2}\,|h'(x)|^{-1/2}$.
Given a cost function c defined on the LFT h, we introduce another Ruelle operator relative to h,

$$\mathbf{R}_{s,h}^{[c]}[f](x) := \frac{c(h)}{D[h](x)^s}\; f \circ h(x). \tag{5}$$

Finally, with an algorithm that uses a set $\mathcal{H}$ of LFT's, one associates two Ruelle operators,

$$\mathbf{H}_s := \sum_{h\in\mathcal{H}} \mathbf{R}_{s,h}, \qquad \mathbf{H}_s^{[c]} := \sum_{h\in\mathcal{H}} \mathbf{R}_{s,h}^{[c]}. \tag{6}$$

The multiplicative property of denominator D, i.e.,

$$D[h\circ g](x) = D[h](g(x))\, D[g](x)$$

is translated by a multiplicative property on Ruelle operators: Given two LFT's, h and g, the Ruelle operator $\mathbf{R}_{s,h\circ g}$ associated with the LFT $h\circ g$ is exactly the operator $\mathbf{R}_{s,g}\circ\mathbf{R}_{s,h}$. More generally, given two sets of LFT's, $\mathcal{L}$ and $\mathcal{K}$ and their Ruelle operators $\mathbf{K}_s, \mathbf{L}_s$, the set $\mathcal{LK}$ is formed of all $h\circ g$ with $h\in\mathcal{L}$ and $g\in\mathcal{K}$, and the Ruelle operator relative to the set $\mathcal{LK}$ is exactly the operator $\mathbf{K}_s\circ\mathbf{L}_s$. In particular, the Ruelle operator relative to the set $\mathcal{H}^\star := \cup_{k\geq 0}\mathcal{H}^k$ is exactly $\sum_{k\geq 0}\mathbf{H}_s^k = (I-\mathbf{H}_s)^{-1}$. This is the quasi–inverse of the Ruelle operator $\mathbf{H}_s$ associated with the set $\mathcal{H}$.

3.4. Ruelle operators and generating functions. We show now how the Ruelle operators intervene in the evaluation of the generating functions of costs $G(s)$. We consider here a Euclidean Algorithm and its set of LFT's $\mathcal{H}$ together with its final set $\mathcal{F}$ defined in Fig.1. The Ruelle operators $\mathbf{H}_s, \mathbf{H}_s^{[c]}, \mathbf{F}_s, \mathbf{F}_s^{[c]}$ relative to $\mathcal{H}$ or $\mathcal{F}$ and defined in (4, 5, 6) will play a central rôle in the analysis.
An execution of the algorithm on the input (u_1, u_0) of Ω performs k external steps of the form $u_0 = m_1u_1 + d_1u_2, \quad \ldots u_{k-1} = m_ku_k + d_ku_{k+1}$ and decomposes the rational (u_1/u_0) as $(u_1/u_0) = h_1\circ h_2\circ\ldots\circ h_k(a)$, where the h_i's are elements of $\mathcal{H}$ (for $i\leq k-1$) or elements of $\mathcal{F}$ (for $i=k$), and a is the last value of the rational. When given an index $i, 1\leq i\leq k$, we consider now three different parts of the LFT $h = h_1\circ h_2\circ\ldots\circ h_k$: the beginning part $b_i(h) := h_1\circ h_2\circ\ldots\circ h_{i-1}$, the ending part $e_i(h) := h_{i+1}\circ h_{i+2}\circ\ldots\circ h_k$, and finally the i-th component h_i. Then the sequence of the rationals (u_{i+1}/u_i) is defined from the relations

$$\frac{u_{i+1}}{u_i} = h_{i+1}\circ h_{i+2}\circ\ldots\circ h_k(a) = e_i(h)(a),$$

and, since u_i and u_{i+1} are coprime, the equality $D[e_i(h)](a) = u_i$ holds.
With an operator $\mathbf{L}_s$, we associate the operator $\Delta\mathbf{L}_s$ defined by

$$\Delta\mathbf{L}_s := \frac{-1}{\log 2}\,\frac{d}{ds}\mathbf{L}_s.$$

When applied to $\mathbf{R}_{s,h}$ defined in (4), the functional Δ is well-suited to the problem since it produces at the numerator the logarithm $\log_2 D[h](x)$. We then introduce the main operator of this work,

$$\mathbf{S}_{s,h} := \sum_{i=1}^{k-1} \Delta \mathbf{R}_{s,e_i(h)} \circ \mathbf{R}^{[c]}_{s,h_i} \circ \mathbf{R}_{s,b_i(h)},$$

and we claim that, when applied to function $f = 1$ and point $x = a$, this operator generates the cost $C(u_1, u_0)$ of the algorithm on input (u_1, u_0), defined in (2),

$$\mathbf{S}_{s,h}[1](a) = \sum_{i=1}^{k-1} \frac{1}{u_0^s}\ c(h_i) \times \log_2 u_i = \sum_{i=1}^{k} \frac{1}{u_0^s}\ c(h_i) \times \log_2 u_i = \frac{1}{u_0^s} C(u_1, u_0).$$

(we replace the upper index k by $k-1$ thanks to equality $u_k = 1$ that holds since u_k is just the gcd of u_0 and u_1). Then, when (u_1, u_0) is a general element of Ω, the LFT h is a general element of the set $\mathcal{H}^\star\mathcal{F}$, so that an alternative expression of the main Dirichlet series F, G defined in (3) holds:

$$F(s) = \sum_{h \in \mathcal{H}^\star\mathcal{F}} \mathbf{R}_{s,h}[1](a), \qquad G(s) = \sum_{h \in \mathcal{H}^\star\mathcal{F}} \mathbf{S}_{s,h}[1](a).$$

When the index i varies in $[1..k-1]$, beginning part $b_i(h)$ is a general element of $\mathcal{H}^\star$, ending part $e_i(h)$ is a general element of $\mathcal{H}^\star\mathcal{F}$, while the i-th component is a general element of $\mathcal{H}$, so that

$$\sum_{h \in \mathcal{H}^\star\mathcal{F}} \mathbf{S}_{s,h} = \Delta\,[\mathbf{F}_s \circ (I - \mathbf{H}_s)^{-1}] \circ \mathbf{H}_s^{[c]} \circ (I - \mathbf{H}_s)^{-1}, \quad \sum_{h \in \mathcal{H}^\star\mathcal{F}} \mathbf{R}_{s,h} = \mathbf{F}_s \circ (I - \mathbf{H}_s)^{-1},$$

where $\mathbf{H}_s, \mathbf{F}_s$ are the Ruelle operators relative to the sets $\mathcal{H}, \mathcal{F}$ used by the algorithm. We finally deduce expressions for the Dirichlet series $F(s), G(s)$ that mainly involve the quasi-inverse of operator $\mathbf{H}_s$:

$$F(s) = \mathbf{F}_s \circ (I - \mathbf{H}_s)^{-1}\ [1](a), \tag{7}$$

$$G(s) \approx \mathbf{F}_s \circ (I - \mathbf{H}_s)^{-1} \circ \Delta \mathbf{H}_s \ \circ\ (I - \mathbf{H}_s)^{-1} \circ \mathbf{H}_s^{[c]} \circ (I - \mathbf{H}_s)^{-1}\ [1](a). \tag{8}$$

(Here, the symbol $\approx$ means that we keep only the main term of the expression, i.e., the one that contains the largest number of occurrences of $(I - \mathbf{H}_s)^{-1}$.)

3.5. Functional Analysis. Here, we consider the following conditions on a set $\mathcal{H}$ of LFT's that will entail all the properties that we need for applying the Tauberian Theorem to the quasi–inverse $(I - \mathbf{H}_s)^{-1}$ of the Ruelle operator.

Conditions $\mathcal{Q}(\mathcal{H})$. *There exist an open disk $\mathcal{V}$ whose closure contains the basic interval $\mathcal{I} := [0, \rho]$, and a real $\alpha < 2$ such that*
(C_1) *For every LFT $h \in \mathcal{H}$, h and $|h'|$ have an analytic continuation on $\mathcal{V}$, and h maps the closure $\bar{\mathcal{V}}$ of disk $\mathcal{V}$ inside $\mathcal{V}$.*

(C_2) *There exists a convenient Banach functional space* $\mathcal{F}(\mathcal{V})$ *formed with analytic functions on* $\mathcal{V}$ *such that each Ruelle operator* $\mathbf{R}_{s,h}$ *(for* $h \in \mathcal{H}$*) acts on* $\mathcal{F}(\mathcal{V})$ *and is compact. Moreover, the series of norms* $\sum_{h\in\mathcal{H}} ||\mathbf{R}_{s,h}||$ *converges on the plane* $\Re(s) > \alpha$.
(C_3) *For* $s = 2$*, the operator* $\mathbf{H}_2$ *is a density transformer: for all* $f \in \mathcal{F}(\mathcal{V})$ *positive on* $\mathcal{V} \cap \mathbf{R}$*, one has* $\int_{\mathcal{I}} \mathbf{H}_2[f](t)dt = \int_{\mathcal{I}} f(t)dt$.
(C_4) *For some integer* A*, the set* $\mathcal{H}$ *contains a subset*

$$\mathcal{D} := \{h \,|\, h(x) = A/(c+x) \quad \text{with integers } c \to \infty\}.$$

If conditions $\mathcal{Q}(\mathcal{H})$ hold, the following main result proves that the quasi–inverse of the Ruelle operator which intervenes in the expressions (7, 8) of generating functions $F(s)$ and $G(s)$ fulfills all the hypotheses of Tauberian Theorem. Moreover, the Dirichlet series $G(s)$ admit a pole at $s = 2$ of order at least three. This order is exactly three, if the Ruelle operator of costs $\mathbf{H}_s^{[c]}$ is regular at $s = 2$. However, the Ruelle operator of costs may have itself a pole at $s = 2$, so that the total order of pole at $s = 2$ becomes four. This implies a general criterion between "fast" algorithms with a log-squared behaviour and "slow" algorithms with a log-cubed behaviour.

Theorem 1. *Let* $(\mathcal{H})$ *be some Euclidean Algorithm that uses a set* $\mathcal{H}$ *of LFTs for which conditions* $\mathcal{Q}(\mathcal{H})$ *hold.*
(a) *Then, the quasi-inverse* $(I-\mathbf{H}_s)^{-1}$ *is analytic on the punctured plane* $\{\Re(s) \geq 2, s \neq 2\}$ *and has a pole of order* 1 *at* $s = 2$*. Near* $s = 2$*, one has, for any function* f *of* $\mathcal{F}(\mathcal{V})$ *positive on* $\mathcal{V} \cap \mathbf{R}$*, and any* $x \in \mathcal{V} \cap \mathbf{R}$*,*

$$(I - \mathbf{H}_s)^{-1}[f](x) \sim \frac{1}{(s-2)} \left(\frac{-1}{\lambda'(2)}\right) \psi_2(x) \int_{\mathcal{I}} f(x)dx, \tag{9}$$

where $\lambda(s)$ *is the dominant eigenvalue of* $\mathbf{H}_s$ *and* ψ_2 *is the dominant eigenfunction of* $\mathbf{H}_2$ *defined by the normalization condition* $\int_{\mathcal{I}} \psi_2(x)dx = 1$.
(b) *Suppose that the Ruelle operator of costs* $\mathbf{H}_s^{[c]}$ *is regular at* $s = 2$*. Then the average bit-complexity of the Euclidean Algorithm on the set of valid inputs of denominator less than* N *is of asymptotic logarithmic–squared order,*

$$\widetilde{B}_N(\mathcal{H}) \sim B_N(\mathcal{H}) \sim A(\mathcal{H}) \log_2^2 N \qquad \text{with} \qquad A(\mathcal{H}) = \frac{\log 2}{h(\mathcal{H})} E_\infty[c].$$

Here $h(\mathcal{H})$ *is the entropy of the dynamical system relative to the algorithm and* $E_\infty[c]$ *denotes the average value of cost* c *related to the LFT* h *when the interval* $\mathcal{I}$ *is endowed with the invariant measure associated with the dominant eigenfunction of* $\mathbf{H}_2$.
(c) *Suppose that the Ruelle operator of costs* $\mathbf{H}_s^{[c]}$ *has a pole of order 1 at* $s = 2$ *and the integral* $I(s) := \int_{\mathcal{I}} \mathbf{H}_s^{[c]}[\psi_2](t)dt$ *satisfies* $I(s)(s-2) \to A$ *for* $s \to 2$*. Then the average bit-complexity of the Euclidean Algorithm on the set of valid inputs of denominator less than* N *is of asymptotic logarithmic–cubed order,*

$$\widetilde{B}_N(\mathcal{H}) \sim B_N(\mathcal{H}) \sim A(\mathcal{H}) \log_2^3 N \qquad \text{with} \qquad A(\mathcal{H}) = \frac{2\log^2 2}{3h(\mathcal{H})} A.$$

Proof. (a) Under conditions (C_1) and (C_2), the Ruelle operator $\mathbf{H}_s$ acts on $\mathcal{F}(\mathcal{V})$ for $\Re(s) > \alpha$ and is compact (even nuclear in the sense of Grothendieck [10,11]). Furthermore, for real values of parameter s, it has positive properties that entail (via Theorems of Perron–Frobenius style due to Krasnoselsky [14]) the existence of dominant spectral objects: there exists a unique dominant eigenvalue $\lambda(s)$ positive, analytic for $s > \alpha$, a dominant eigenfunction denoted by ψ_s, and a dominant projector e_s. Under normalization condition $e_s[\psi_s] = 1$, these last two objects are unique too. Then, the compacity entails the existence of a spectral gap between the dominant eigenvalue and the remainder of the spectrum, that separates the operator $\mathbf{H}_s$ and the quasi-inverse $(I - \mathbf{H}_s)^{-1}$ in two parts : the "part" relative to the dominant eigenvalue, and the "part" relative to the remainder of the spectrum. Under conditions (C_3), the operator $\mathbf{H}_2$ is a density transformer, so that $\lambda(2) = 1$ and $e_2[f] = \int_{\mathcal{I}} f(t)dt$. On the other hand, condition (C_4) implies that the operator $\mathbf{H}_s$ has no eigenvalue equal to 1 on the line $\Re(s) = 2, s \neq 2$.
(b) Here, the Dirichlet series $G(s)$ has a triple pole at $s = 2$, and near $s = 2$

$$G(s) \sim (\frac{-1}{\lambda'(2)})^3 \; \frac{\mathbf{F}_2[\psi_2](a)}{(s-2)^3} \; \Big(\int_{\mathcal{I}} \Delta\mathbf{H}_2[\psi_2](t)dt\Big) \; \Big(\int_{\mathcal{I}} \mathbf{H}_2^{[c]}[\psi_2](t)dt\Big).$$

Both integrals are easily transformed. The first one

$$\int_{\mathcal{I}} \Delta\mathbf{H}_2[\psi_2](t)dt = \int_{\mathcal{I}} |\log_2 t|\, \psi_2(t)dt$$

equals $-\lambda'(2)/\log 2$ and coincides with $h(\mathcal{H})/(2\log 2)$, where $h(\mathcal{H})$ is the entropy of the dynamical system. The second integral deals with the cost $c(h)$ of the LFT

$$\int_{\mathcal{I}} \mathbf{H}_2^{[c]}[\psi_2](t)dt = \sum_{h\in\mathcal{H}} c(h) \int_{h(\mathcal{I})} \psi_2(t)dt$$

and coincides with the average value of cost c when the interval $\mathcal{I}$ is endowed with the invariant measure $\psi_2(t)dt$. This average value is denoted by $E_\infty[c]$ and it is a constant of Khinchin's type.

(c) Here, the Dirichlet series $G(s)$ has a pole of order four at $s = 2$.

4 Average-Bit Complexity of the Algorithms

We now come back to the analysis of the five algorithms, and we study successively the fast algorithms, then the slow ones.

4.1. The fast algorithms. We study here three algorithms: the Classical Algorithm $(\mathcal{G})$, the Centered Algorithm $(\mathcal{K})$ and the Binary Algorithm $(\mathcal{B})$. We begin by the first two, that constitute "easy cases".
The Classical Algorithm and the Centered Algorithm. We first consider the sets $\mathcal{G}, \mathcal{K}$ relative to the Classical Algorithm or the Centered Algorithm. There exists,

in both cases, an open disk $\mathcal{V}$ whose diameter *strictly* contains the basic interval $\mathcal{I}$, such that each LFT h maps $\bar{\mathcal{V}}$ strictly inside itself. A convenient functional space is then the set $\mathcal{A}_\infty(\mathcal{V})$ formed with functions f that are analytic on $\mathcal{V}$ and continuous on $\bar{\mathcal{V}}$. Endowed with the sup-norm, this set is a Banach space and each Ruelle operator $\mathbf{R}_{s,h}$ acts on this set and is compact. The series of (C_2) is convergent for $\Re(s) > 1$. Moreover, at $s = 2$, the Ruelle operators are density transformers. The (normalized) invariant functions ψ_2 are explicit in the classical case and in the centered case,

$$\frac{1}{\log 2}\frac{1}{1+x}, \qquad \frac{1}{\log\phi}[\frac{1}{\phi+x}+\frac{1}{\phi^2-x}] \qquad \text{with} \quad \phi = \frac{1+\sqrt{5}}{2}.$$

Now, we can apply Theorem 2, version (b). Here, since invariant function ψ_2 is explicit, the same is true for the entropy that equals $\pi^2/(6\log 2)$ in the classical case and equals $\pi^2/(6\log\phi)$ in the centered case. Moreover, the average value $E_\infty[c]$ is also easily computed from explicit forms of cost c, of invariant function ψ_2 and distribution function F_2.

The Binary Algorithm. The set $\mathcal{B}$ of LFT's relative to the Binary algorithm is more difficult to deal with (see [3], [23]). First, it is not possible to find an open disk whose diameter contains the basic interval $\mathcal{I} := [0,1]$ and on which all the LFT's are analytic. The reason is that the sequence of poles of LFT's is of the form $x = -m/2^s$ and has an accumulation point at $x = 0$. We choose for $\mathcal{V}$ an open disk of diameter $]0,\alpha[$ with $1 < \alpha < 2$, and a convenient functional space is then the Hardy space of order two relative to $\mathcal{V}$. It is denoted by $\mathcal{H}^2(\mathcal{V})$ and is formed with all functions f analytic inside $\mathcal{V}$ and such that $|f|^2$ is integrable along the frontier of $\mathcal{V}$. Each Ruelle operator $\mathbf{R}_{s,h}$ acts on this set and is compact. The series of (C_2) is convergent for $\Re(s) > (3/2)$.

Now, we can apply Theorem 2, version (b) as previously. However, the dynamical system with which the algorithm is associated is now more complex, since it is a random dynamical system. The reason is that the pseudo-division is related to dyadic valuation, so that the binary continued fraction expansion is only defined for rationals numbers. However, one can define random binary continued fraction for real numbers when choosing at random the dyadic valuation k of a real number, according to the law $\Pr[k = d] = 2^{-d}$ (for $d \geq 1$) that extends the natural law on even integers. In this manner, we choose the LFT of determinant 2^k with probability 2^{-k}, and, for LFT's of determinant 2^k, the quantity

$$D[h](x)^{-2}\ f\circ h(x)dx = 2^{-k}|h'(x)|\ f\circ h(x)dx$$

represents exactly a random change of variables. Then, the Ruelle operator can be viewed as the transfer operator relative to this random dynamical system and for $s = 2$ it is a (random) density transformer. Furthermore, even if the invariant eigenfunction ψ_2, the distribution function F_2 are no more explicit, the entropy and the average value $E_\infty[c]$ can be expressed as functions of F_2 and ψ_2.

Theorem 2. *The average bit-complexities of the Classical Algorithm ($\mathcal{G}$), the Centered Algorithm ($\mathcal{K}$) and the Binary Algorithm ($\mathcal{B}$) on the set of valid inputs of denominator less than N are of asymptotic logarithmic–squared order. They*

satisfy, for $\mathcal{H} \in \{\mathcal{G}, \mathcal{K}, \mathcal{B}\}$,

$$\widetilde{B}_N(\mathcal{H}) \sim B_N(\mathcal{H}) \sim \; A(\mathcal{H}) \log_2^2 N \qquad \text{with} \qquad A(\mathcal{H}) = \frac{\log 2}{h(\mathcal{H})} E_\infty[c]$$

Here $h(\mathcal{H})$ *is the entropy of the dynamical system relative to the algorithm and* $E_\infty[c]$ *denotes the average value of cost* c *related to the LFT* h *when the interval* $\mathcal{I}$ *is endowed with the invariant measure. In the classical case* ($\mathcal{G}$)*, the cost* $c(h)$ *equals* $\ell(m) + 2$ *where* $\ell(m)$ *is the number of bits of digit* m*, and*

$$A(\mathcal{G}) = \frac{6 \log^2 2}{\pi^2} \; \big[2 + \frac{1}{\log 2} \log \prod_{k=0}^{\infty} (1 + \frac{1}{2^k})\big];$$

in the centered case ($\mathcal{K}$)*, the cost* $c(h)$ *equals* $\ell(m) + 2 + (1 - \varepsilon)/2$ *where* $\ell(m)$ *is the number of bits of digit* m*, and* $\varepsilon = \pm 1$ *the sign used, so that*

$$A(\mathcal{K}) = \frac{6 \log \phi \log 2}{\pi^2} \big[3 + \frac{\log 2}{\log \phi} + \frac{1}{\log \phi} \log \prod_{k=3}^{\infty} \frac{(2^k - 1)\phi^2 + 2\phi}{(2^k - 1)\phi^2 - 2}\big].$$

In the binary case ($\mathcal{B}$)*, the cost* $c(h)$ *equals* $b(m) + 2$ *where* $b(m)$ *is the number of ones in the binary expansion of digit* m*, and*

$$A(\mathcal{B}) = \frac{2 \log 2}{\pi^2 \, \psi_2(1)} \; \big[1 + \sum_{m \text{ odd} \geq 1} \frac{1}{2^{\ell(m)}} F_2(\frac{1}{m})\big],$$

where $\psi_2(t)dt$ *is the invariant measure,* F_2 *its distribution function, and, as before,* $\ell(m)$ *is the number of bits of integer* m*.*

4.2. The slow algorithms. We study now the Subtractive Algorithm ($\mathcal{T}$) and the By-Excess Algorithm ($\mathcal{L}$). In these cases, the analysis requires a special twist that takes its inspiration from the study of intermittency phenomena in physical systems that was introduced by Bowen [2] and is nicely exposed in a paper of Prellberg and Slawny [16]. Even if the Subtractive Algorithm can be directly analyzed, it is clearer to describe both analyses inside the same framework.
First, we remark that each (internal) step of the Subtractive algorithm may use two LFT's $p(x) := x/(1 + x), q(x) := 1/(1 + x)$, depending if the subtraction is followed or not by an exchange.
In both cases ($\mathcal{T}, \mathcal{L}$), the set of LFTs $\mathcal{H}$ does not fulfill conditions $\mathcal{Q}(\mathcal{H})$ since it contains one "bad" LFT which possesses an indifferent point, i.e., a fixed point where the absolute value of the derivative equals 1: this is $p : x \to 1/(2 - x)$ for $\mathcal{L}$ and $p : x \to x/(1 + x)$ for $\mathcal{T}$. In this case, it is not possible to find an open disk $\mathcal{V}$ that contains the basic interval $\mathcal{I}$ and such that the LFT p maps $\bar{\mathcal{V}}$ inside $\mathcal{V}$. However, the remainder of the set, i.e., the set $\mathcal{H} \setminus \{p\}$ is well-behaved, so that we adapt the *method of inducing* that originates from dynamical systems theory. The main idea is to consider a sequence of p to be followed by a good LFT q that belong to $\mathcal{H} \setminus \{p\}$. This is always possible since final set $\mathcal{F}$ does not

contain p. In that way, instead of set $\mathcal{H}$, we use set $\mathcal{M} := p^\star\,[\mathcal{H} \setminus \{p\}]$ that now fulfills conditions $\mathcal{Q}(\mathcal{M})$.
A sequence of iterations that uses an element $p^k q$ of $\mathcal{M}$ with $k \geq 0$ and $q \neq p$ deals with successive integers u_i (for $1 \leq i \leq k$) that are slowly decreasing, so that the global cost of the sequence of iterations is well-approximated by $k \log_2 u_0$. Finally, one can apply Theorem 2, version (c), to the set $\mathcal{M}$, and the integral relative to the Ruelle operator of costs $\int_{\mathcal{I}} \mathbf{M}_s^{[c]}[\psi_2](t)dt$ is a divergent series at $s = 2$ with residue A equals $1/(2\log 2)$ in the $\mathcal{L}$–case and $1/\log 2$ in the $\mathcal{T}$–case.

Theorem 3. *The average bit–complexities of the By-Excess Algorithm ($\mathcal{L}$) and the Subtractive Algorithm ($\mathcal{T}$) on the set of valid inputs of denominator less than N are of asymptotic log–cubed order. They all satisfy*

$$\widetilde{B}_N(\mathcal{H}) \sim B_N(\mathcal{H}) \sim \; A(\mathcal{H}) \log^3 N \quad \textit{with } A(\mathcal{T}) = \frac{2\log^2 2}{\pi^2}, \quad A(\mathcal{L}) = \frac{\log^2 2}{\pi^2}.$$

References

1. Bedford, T., Keane, M., and Series, C., Eds. *Ergodic Theory, Symbolic Dynamics and Hyperbolic Spaces*, Oxford University Press, 1991.
2. Bowen, R. Invariant measures for Markov maps of the interval, *Commun. Math. Phys.* 69 (1979) 1–17.
3. Brent, R.P. Analysis of the binary Euclidean algorithm, Algorithms and Complexity, New directions and recent results, ed. by J.F. Traub, Academic Press 1976, pp 321–355
4. Daudé, H., Flajolet, P., and Vallée, B. An average-case analysis of the Gaussian algorithm for lattice reduction, *Combinatorics, Probability and Computing* (1997) 6 397–433
5. Delange, H. Généralisation du Théorème d'Ikehara, *Ann. Sc. ENS, (1954) 71, pp 213–242.*
6. Dixon, J. D. The number of steps in the Euclidean algorithm, *Journal of Number Theory 2* (1970), 414–422.
7. Flajolet, P. Analytic analysis of algorithms, In Proceedings of the 19th International Colloquium "Automata, Languages and Programming", Vienna, July 1992, W. Kuich, editor, Lecture Notes in Computer Science 623, pp 186–210
8. Flajolet, P. and Sedgewick, R. Analytic Combinatorics, Book in preparation (1999), see also INRIA Research Reports 1888, 2026, 2376, 2956.
9. Flajolet, P., and Vallée, B. Continued fraction Algorithms, Functional operators and Structure constants, *Theoretical Computer Science* 194 (1998), 1–34.
10. Grothendieck, A. Produits tensoriels topologiques et espaces nucléaires, *Mem. Am. Math. Soc.* 16 (1955)
11. Grothendieck, A. La théorie de Fredholm, *Bull. Soc. Math. France* 84 pp 319-384.
12. Heilbronn, H. On the average length of a class of continued fractions, Number Theory and Analysis, ed. by P. Turan, New-York, Plenum, 1969, pp 87-96.
13. Hensley, D. The number of steps in the Euclidean algorithm, *Journal of Number Theory* 49, 2 (1994), 142–182.
14. Krasnoselskii, M. *Positive solutions of operator equations*, P. Noordhoff, Groningen, 1964.

15. Mayer, D. H. Continued fractions and related transformations, In *Ergodic Theory, Symbolic Dynamics and Hyperbolic Spaces*, T. Bedford, M. Keane, and C. Series, Eds. Oxford University Press, 1991, pp. 175–222.
16. Prellberg, T. and Slawny, J. Maps of intervals with Indifferent fixed points: Thermodynamic formalism and Phase transitions. *Journal of Statistical Physics* 66 (1992) 503-514
17. Rieger, G. J. Uber die mittlere Schrittazahl bei Divisionalgorithmen, *Math. Nachr.* (1978) pp 157-180
18. Ruelle, D. *Thermodynamic formalism,* Addison Wesley (1978)
19. Ruelle, D. *Dynamical Zeta Functions for Piecewise Monotone Maps of the Interval*, vol. 4 of *CRM Monograph Series*, American Mathematical Society, Providence, 1994.
20. Tenenbaum, G. *Introduction à la théorie analytique des nombres*, vol. 13. Institut Élie Cartan, Nancy, France, 1990.
21. Vallée, B. Opérateurs de Ruelle-Mayer généralisés et analyse des algorithmes d'Euclide et de Gauss, *Acta Arithmetica* 81.2 (1997) 101–144.
22. Vallée, B. Fractions continues à contraintes périodiques, *Journal of Number Theory* 72 (1998) pp 183–235.
23. Vallée, B. Dynamics of the Binary Euclidean Algorithm: Functional Analysis and Operators., *Algorithmica* (1998) vol 22 (4) pp 660–685.
24. Vallée, B. A Unifying Framework for the Analysis of a Class of Euclidean Algorithms., To appear in the proceedings of LATIN'2000, LNCS
25. Vardi, I. Continued fractions, Preprint, chapter of a book in preparation.
26. Yao, A.C., and Knuth, D.E. Analysis of the subtractive algorithm for greatest common divisors. *Proc. Nat. Acad. Sc. USA* 72 (1975) pp 4720-4722.

Planar Maps and Airy Phenomena

Cyril Banderier[1], Philippe Flajolet[1], Gilles Schaeffer[2], and Michèle Soria[3]

[1] Algorithms Project, INRIA, Rocquencourt, 78150 Le Chesnay (France)
[2] Loria, CNRS, Campus scientifique, B.P. 239, 54506 Vandœuvre-lès-Nancy (France)
[3] Lip6, Université Paris 6, 8 rue du Capitaine Scott, 75005 Paris (France).

Abstract. A considerable number of asymptotic distributions arising in random combinatorics and analysis of algorithms are of the exponential-quadratic type (e^{-x^2}), that is, Gaussian. We exhibit here a new class of "universal" phenomena that are of the exponential-cubic type (e^{ix^3}), corresponding to nonstandard distributions that involve the Airy function. Such Airy phenomena are expected to be found in a number of applications, when confluences of critical points and singularities occur. About a dozen classes of planar maps are treated in this way, leading to the occurrence of a common Airy distribution that describes the sizes of cores and of largest (multi)connected components. Consequences include the analysis and fine optimization of random generation algorithms for multiply connected planar graphs.

Maps are planar graphs presented together with an embedding in the plane, and as such, they model the topology of many geometric arrangements in the plane and in low dimensions (e.g., 3-dimensional convex polyhedra). This paper concerns itself with the statistical properties of random maps, *i.e.*, the question of what such a random map typically looks like. We focus here on connectivity issues, with the specific goal of finely characterizing the size of the highly connected "core" of a random map.

The bases of an enumerative theory of maps have been laid down by Tutte [22] in the 1960's, in an attempt to attack the four-colour conjecture. The present paper builds upon Tutte's results and upon the detailed yet partial analyses of largest components given by Bender, Richmond, Wormald, and Gao [2,11]. We establish the common occurrence of a new probability distribution, the "map–Airy distribution", that precisely quantifies the sizes of cores in about a dozen varieties of maps, including general maps, triangulations, 2-connected maps, etc. As a corollary, we are able to improve on the complexity of the best known random samplers for multiply connected planar graphs and convex polyhedra [19].

The analysis that we introduce is largely based on a method of "coalescing saddle points" that was perfected in the 1950's by applied mathematicians [3,24,1] and has found scattered applications in statistical physics and the study of phase transitions [16]. However, this method does not appear to have been employed so far in the field of random combinatorics. We claim some generality for the approach proposed here on at least two counts. First, a number of enumerative problems are known to be of the "Lagrangean type", being

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 388–402, 2000.
© Springer-Verlag Berlin Heidelberg 2000

related to the Lagrange inversion theorem and its associated combinatorics. The classical saddle point method is then instrumental in providing asymptotics of simpler problems. However, confluence of saddle points is a stumbling block of the basic method. As we show here, planar maps are precisely instances of this special situation. Next, the method extends to the analysis of a new composition scheme. Indeed, it is known, in the realm of analytic combinatorics, that asymptotic properties of random structures are closely related to singular exponents of counting generating functions. For "most" recursive objects the exponent is $\frac{1}{2}$ and the probabilistic phenomena are described by classical laws, like Gaussian, exponential, or Poisson. Methods of the paper permit us to quantify distributions associated with singular exponents $\frac{3}{2}$ present in maps and unrooted trees and leading to Airy laws.

Very roughly, the classical saddle point method gives rise to probabilistic and asymptotic phenomena that are in the scale of $n^{1/2}$ and the analytic approximations are in the form of an "exponential-quadratic" (e^{-x^2}) corresponding to Gaussian laws. The coalescent saddle-point method presented here gives rise to phenomena in the scale of $n^{1/3}$, with analytic approximations of the "exponential-cubic type" (e^{ix^3}), which, as we shall explain, is conducive to Airy laws. The Airy phenomena that we uncover in random combinatorics should thus be expected to be of a fair degree of universality. To support this claim, here are scattered occurrences of what we recognize as Airy phenomena in the perspective of this paper: the emergence of first cycles and of the giant component in the Erdös-Rényi graph model [8,13], the enumeration of random forests of unrooted trees [14], clustering formation in the construction of linear probing hash tables [10], the area under excursions and the cumulative storage cost of dynamically varying stacks [15], the area of certain polyominoes [7], path length in combinatorial tree models [21], and (we conjecture) the threshold phenomena involved in the celebrated random 2-SAT problem [4]. We propose to elaborate on these connections in future papers.

Plan of the paper. Basics of maps are introduced in Section 1 where the Airy distribution is presented. The enumerative theory can be developed along two parallel lines, one Lagrangean, the other based on singularity analysis. We first approach the analysis of core size via the Lagrangean framework and variations on the saddle point method: a fine analysis of the geometry of associated complex curves is shown to open access to the size of the core, with the Airy distribution arising from double or "nearby" saddles (Section 2); a refined analysis based on the method of coalescent saddle points then enables us to quantify the distribution of core size over a wide range with precise large deviation estimates (Section 3). The method applies to about a dozen of types of planar maps, it provides a precise quantification of largest components, with consequences on the random generation of highly connected planar graphs (Section 4). Finally, we show that the very same Airy law is bound to occur in any instance of a general composition scheme of analytic combinatorics (Section 5).

1 Basics of Maps

A *map* is a planar graph given together with an embedding in the plane considered up to continuous deformations. Following Tutte, we consider *rooted* maps, that is, maps with an oriented edge called the *root*—this simplifies this analysis without essentially affecting statistical properties (see [17] and Section 4). Generically, we take $\mathcal{M}$ and $\mathcal{C}$ to be two classes of maps, with $\mathcal{M}_n$, $\mathcal{C}_n$ the subsets of elements of size n (typically elements with $n+1$ edges). Here, $\mathcal{C}$ is always a subset of $\mathcal{M}$ that satisfies additional properties (*e.g.* higher connectivity). The elements of $\mathcal{M}$ are then called the "basic maps" and the elements of $\mathcal{C}$ are called the "core-maps". We define the *core-size* of a map $m \in \mathcal{M}$ as the size of the largest $\mathcal{C}$–component of m that contains the root of m. As a pilot example, we shall specialize the basic maps $\mathcal{M}_n$ to be the class of nonseparable maps (*i.e.*, 2-connected loopless maps) with $n+1$ edges and $\mathcal{C}_k$ to be the set of 3-connected maps with $k+1$ edges.

Our major objective is to characterize the probabilistic properties of core-size of a random element of $\mathcal{M}_n$, that is, of a random map of size n, when all elements are taken equally likely. Core-size then becomes a random variable X_n defined on $\mathcal{M}_n$. In essence, the pilot example thus deals with 3-connectivity in random 2-connected maps. The paradigm that we illustrate by a particular example is in fact of considerable generality as can be seen from Sections 4, 5 below.

The physics of maps. From earlier works [2,11,18], it is known that a random map of $\mathcal{M}_n$ has with high probability a core that is either "small" (roughly of size $k = O(1)$) or "large" (being $\Theta(n)$). The probability distribution $\Pr(X_n = k)$ thus has two distinct modes. The small region (say $k = o(n)$) has been well quantified by previous authors, see [2,11,18]: a fraction $p_s = \frac{65}{81}$ of the probability mass is concentrated there. The large region is also known from these authors to have probability mass $p_\ell = 1 - p_s = \frac{16}{81}$ concentrated around $\alpha_0 n$ with $\alpha_0 = \frac{1}{3}$ but this region has been much less explored as it poses specific analytical difficulties. Our results precisely characterize what happens in terms of an Airy distribution.

The Airy function $\mathrm{Ai}(z)$, as introduced by the Royal Astronomer Sir George Bidell Airy, is a solution of the equation $y'' - zy = 0$ that can be defined by a variety of integral or power series representations including [23]:

$$\mathrm{Ai}(z) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} e^{i(zt+t^3/3)}\,dt = \frac{1}{\pi 3^{2/3}}\sum_{n=0}^{\infty}\frac{\Gamma((n+1)/3)}{n!}\sin\frac{2(n+1)\pi}{3}\left(3^{1/3}x\right)^n. \tag{1}$$

Equipped with this definition, we present the main character of the paper.

Definition 1. *The (standard) "map–Airy" distribution is the probability distribution whose density is*

$$\mathcal{A}(x) = 2\exp\left(-\frac{2}{3}x^3\right)\left(x\mathrm{Ai}(x^2) - \mathrm{Ai}'(x^2)\right).$$

The "map–Airy" distribution of parameter c is defined by its density, $c\mathcal{A}(cx)$.

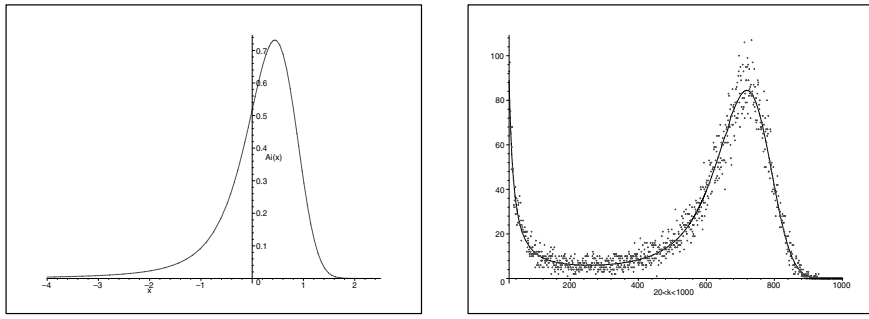


Fig. 1. (i) The map Airy distribution. (ii) Observed frequencies of core-sizes $k \in [20, 1000]$ in 100,000 random maps of size 2000, against predictions of Thms 3, 4.

Note the nonobvious fact that the map–Airy distribution is a probability distribution, *i.e.*, $\int_{\mathbb{R}} \mathcal{A}(x)\, dx = 1$, which can be checked by Mellin transform techniques. An unusual feature is the fact that the tails are extremely asymmetric: $\mathcal{A}(x) = O\left(|x|^{-5/2}\right)$, as $x \to -\infty$, and $\mathcal{A}(x) = O\left(x^{1/2} \exp\left(-\frac{4}{3}x^3\right)\right)$, as $x \to +\infty$. We shall find that the size of the core (conditioned upon the large region) is described asymptotically by an Airy law of this type; see Figure 1.

The combinatorics of maps. Let M_n and C_k be the cardinalities of $\mathcal{M}_n$ and $\mathcal{C}_k$. The *generating functions* of $\mathcal{M}$ and $\mathcal{C}$ are respectively defined by

$$M(z) := \sum_{n \geqslant 1} M_n z^n, \qquad \text{and} \qquad C(z) := \sum_{k \geqslant 1} C_k z^k.$$

(i) *Root-face decomposition.* As shown by Tutte, there results from a root-face decomposition and from the quadratic method [12, Sec. 2.9] that the generating function of $M(z)$ is Lagrangean, which means that it can be parametrized by a system of the form

$$M(z) = \psi(L(z)) \quad \text{where} \quad L(z) = z\phi(L(z)), \tag{2}$$

for two power series ψ, ϕ, with L being determined implicitly by ϕ. For nonseparable maps, we have $\phi(y) = (1+y)^3, \quad \psi(y) = y(1-y)$. There results from the form (2) and from the Lagrange inversion theorem [12] an explicit form for the coefficients of $M(z)$, namely,

$$M_n \equiv [z^n]M(z) = \frac{1}{n}[y^n]\psi'(y)\phi(y)^n, \tag{3}$$

where $[z^n]f(z)$ denotes the coefficient of z^n in the series expansion of $f(z)$. For nonseparable maps, this instantiates to $M_n \equiv [z^n]M(z) = \frac{4(3n)!}{n!(2n+2)!}$.

(ii) *Substitution decomposition.* As shown again by Tutte, maps satisfy additionally relations of the "substitution type": one has: $M(z) = \left(z + \frac{2M(z)^2}{1+M(z)}\right) + C(M(z))$, meaning that each map (left part) either has no core (right part, the

first term) or is formed of a nondegenerate core in which maps are substituted (right part, the second term). This equation effectively gives access to the exact enumeration of objects of type $\mathcal{C}$ that are more "complex", *i.e.*, more highly connected than the initial maps of $\mathcal{M}$.

Our interest lies in the probability $\Pr(X_n = k)$ that a map of $\mathcal{M}_n$ has a core with $k+1$ edges. Let $\mathcal{M}_{n,k}$ be the set of maps with this property; we define the *bivariate generating function* $M(z,u) = \sum_{n,k} M_{n,k} u^k z^n$, with $M_{n,k} = \operatorname{card}(\mathcal{M}_{n,k})$. Tutte proved the following refinement: $M(z,u) = C(uM(z))$. This determines the *probability distribution of the core-size*:

$$\Pr(X_n = k) = \frac{C_k\,[z^n]M(z)^k}{M_n},\ [z^n]M(z)^k = \frac{k}{n}[y^{n-1}]y\psi'(y)\psi(y)^{k-1}\phi(y)^n, \quad (4)$$

where the second equality results from Lagrange inversion.

All the involved generating functions are algebraic functions leading to complicated alternating binomial sums expressing $\Pr(X_n = k)$. The exponential cancellations involved are however not tractable in this elementary way, and complex asymptotic methods must be resorted to.

The asymptotics of maps. There are here two sides to the coin: one evoked now and explored further in Section 5 relies on singularity analysis [9], a method that establishes a general correspondence between the expansion of a generating function at a singularity and the asymptotic form of its coefficients; the other discussed in the next two sections makes use of the power forms provided by the Lagrange inversion theorem that can be exploited asymptotically by the saddle point method.

An implicitly defined function like $L(z)$ in (2) has a singularity of the square-root type $L(z) = \tau - c(1-z/\rho)^{1/2} + O(1-z/\rho)$, where the singularity ρ and the singular value τ are determined by the equations $\tau\phi'(\tau) - \phi(\tau) = 0$, $\rho = \frac{\tau}{\phi(\tau)}$. This expansion yields the singular expansion of the generating function of maps,

$$M(z) = \psi(\tau) - a(1-z/\rho) + b(1-z/\rho)^{3/2} + O((1-z/\rho)^2) \quad (5)$$

(in all known map-related cases, one has $\psi'(\tau) = 0$ which induces the singular exponent of $3/2$). According to singularity analysis (or the Darboux-Pólya method), this last expansion entails

$$M_n \sim \frac{3b}{4\sqrt{\pi}}\frac{\rho^{-n}}{n^{5/2}},\ \tau = \frac{1}{2},\ \rho = \frac{4}{27},\ M_n^{(\text{nonsep.})} \sim \frac{\sqrt{3}}{2\sqrt{\pi}}\left(\frac{27}{4}\right)^n n^{-5/2} \quad (6)$$

Finally, this approach also yields by direct inversion the asymptotic number of core maps: $C(z)$ has a singularity at $\psi(\tau)$ and singular exponent $\frac{3}{2}$. We have

$$C(z) = c_0 - a'(1-z/\psi(\tau)) + b'(1-z/\psi(\tau))^{3/2} + O((1-z/\psi(\tau))^2), \quad (7)$$

$$C_k = [z^k]C(z) \sim \frac{3b'}{4\sqrt{\pi}}\psi(\tau)^{-k}k^{-5/2},\ C_k^{(3-\text{conn.})} \sim \frac{8}{243\sqrt{\pi}}4^k k^{-5/2} \quad (8)$$

The foregoing discussion is then conveniently summarized by a statement that constitutes the starting point of our analysis.

Proposition 1. *The distribution of the size of the core in nonseparable maps is characterized by Eq. (4), where the core map counts C_k are determined asymptotically by (8). The basic maps in $\mathcal{M}_n$ are enumerated exactly by (3) and asymptotically by (6).*

2 Two Saddles

The probability distribution of core-size in maps is determined by Proposition 1, especially by Equation (4). What is needed is a way to estimate $[z^n]M(z)^k$. The approach starts from the contour integral representation deriving from Cauchy's coefficient formula,

$$[z^n]M^k(z) = \frac{k}{n}\frac{1}{2i\pi}\int_\Gamma z(\psi(z)^k)'\phi(z)^n\frac{dz}{z^{n+1}} = \frac{k}{n}\frac{1}{2i\pi}\int_\Gamma G(z)\,\psi(z)^k\,(\phi(z)/z)^n\;dz \tag{9}$$

where Γ is a contour encircling the origin anticlockwise and $G(z) = \psi'(z)/\psi(z) = (1-2z)/(z(1-z))$.

In simpler cases, integrals over complex contours involving large powers are amenable to the basic saddle point method. The idea consists in deforming the contour Γ in the complex plane, this, in order to have it cross a saddle point of the integrand (*i.e.*, a zero of the derivative) and to take advantage of concentration of the integral near the saddle point. Then local expansions are of the "exponential quadratic" type and the (real-variable) Laplace method permits one to estimate the integral asymptotically [5].

For the problem at hand, there are two saddle points, given by the equation $\frac{\partial}{\partial z}(k\ln\psi + n\ln(\phi/z)) = 0$:

$$z_+(n,k) = \frac{1}{2} \quad \text{and} \quad z_-(n,k) = \frac{n-k}{n+k}.$$

The basic saddle point method applies when these two points are distinct, that is, as long as k/n is "far away" from $\frac{1}{3}$. This corresponds to the situation already well-known from the works of [2,11,18]. The "interesting" region is however when $k = n/3$ and when k is close to $n/3$ in the scale of $n^{2/3}$. In that case, the basic version of the saddle point method is no longer applicable. This is precisely where we fit in: we prove that a detailed examination of the analytic geometry of the saddle points in conjunction with suitable integration contours "captures" the major contributions and leads to a precise quantification of core-size in random maps.

Distinct saddles When k is far enough from $n/3$, one of the saddle points is nearer to the origin and predominates. In that case, the basic method applies using a contour that is a circle centered at the origin, passing through the dominant saddle point. This corresponds to the already known results of [2,11] supplemented by [18].

Theorem 1 (Tails and distinct saddles [11]). *Let $\lambda(n)$ be an arbitrary function with $\lambda(n) \to +\infty$ and $\lambda(n) = o(n^{1/3})$. Then, the probability distribution of*

the core of random element of $\mathcal{M}_n$ satisfies

$$\Pr(X_n = k) \sim \frac{32}{243\sqrt{\pi}} \cdot \frac{n^{5/2}}{k^{3/2}(n-3k)^{5/2}}, \quad \textit{uniformly for } \lambda(n) < k < \tfrac{n}{3} - n^{2/3}\lambda(n)$$
$$\Pr(X_n = k) = O\left(\exp(-n(k/n - 1/3)^3)\right), \quad \textit{uniformly for } k > \tfrac{n}{3} + n^{2/3}\lambda(n).$$

Proof (Sketch). The left tail ($n < 3k$) corresponds to the saddle point $z_+ = \frac{1}{2}$ that is dominant (*i.e.*, nearer to the origin and providing the major asymptotic contribution). The right tail ($n > 3k$) has $z_- = (n-k)/(n+k)$ dominating. In each case, the basic saddle point method applies.

A double saddle Here we attack directly the analysis of the "center" of the distribution, that is, the case where $n = 3k$ exactly. Then, the saddle points become equal: $z_- = z_+$. This case serves to introduce with minimal apparatus the enhancements that need to be brought to the basic saddle point method. Observe that the complete confluence of the saddle points precludes the use of "exponential-quadratic" approximations and the problem becomes of an "exponential cubic" type. (See also [2] for a partial discussion of this case based on a method of Van der Corput.)

Theorem 2 (Central part and a double saddle). *The probability distribution of the core of random element of $\mathcal{M}_n$ satisfies, when $n = 3k$,*

$$\Pr(X_{3k} = k) = \frac{4}{27}\frac{\Gamma(2/3)}{3^{1/6}\pi}k^{-2/3}\left(1 + O((\ln(k))^4 k^{-1/3})\right), \quad \frac{4}{27}\frac{\Gamma(2/3)}{3^{1/6}\pi} \approx 0.0531.$$

Proof. When $n = 3k$, equation (9) becomes

$$[z^{3k}]M^k(z) = \frac{1}{6i\pi}\int_\Gamma G(z)P(z)^k dz, \tag{10}$$

where $P(z) := \psi\phi^3/z^3 = (1-z)(1+z)^9/z^2$ and the "kernel" $\ln(P)$ (together with P, P^k) now has a double saddle point at $\tau = z_- = z_+ = \frac{1}{2}$, sometimes called a "monkey saddle", *viz.*, a saddle with places for two legs and a tail. The idea consists in choosing a contour that is no longer a circle centered at the origin, but, rather, approaches the real axis at an angle. Specifically, the integration path Γ consists of the following: the part Γ_0 of a circle centered at 0 from which a small arc is taken out, joining with two (small) segments Δ_1, Δ_2 of length δ that intersect at $\frac{1}{2}$ at an angle of $\pm 2\pi/3$.

We shall adopt a value of δ satisfying two conflicting requirements,

$$n\delta^3 \to \infty, \qquad n\delta^4 \to 0, \quad \text{specifically} \quad \delta = (\ln n)n^{-1/3}. \tag{11}$$

The kernel $\ln(P)$ has a double saddle point in τ, meaning that its local expansion is of the cubic type:

$$\ln(P(z)) = \ln(P(\tau)) - d(z-\tau)^3 + O((z-\tau)^4), \qquad d = \frac{64}{9}.$$

The geometry of the level curves of the kernel shows that the contribution $\mathcal{E}_0$ along Γ_0 to the integral in (10) is bounded by a constant times the value of $P(z)^k$ at the endpoints of Γ_0. This contribution then satisfies

$$\mathcal{E}_0 \equiv \int_{\Gamma_0} G(z)\exp[k\ln(P(z))]\,dz = O(P(\tau)^k \exp(-kd\delta^3)),$$

which, given the constraints on δ (condition $n\delta^3 \to \infty$ in (11)) is exponentially small.

The contribution $\mathcal{E}_{1,2}$ along $\Delta_1 \cup \Delta_2$ to the integral in (10) provides the dominant contribution and is estimated next by a local analysis of P^k for values of z near τ. Set $u = z - \tau$. The condition $n\delta^4 \to 0$ in (11) implies that terms of order 4 and higher do not matter asymptotically, and a simple calculation, using the fact that $G(\tau + u) = -8u + O(u^2)$, yields

$$\mathcal{E}_{1,2} \equiv \int_{\Delta_1 \cup \Delta_2} G(z) \exp[k \ln(P(z))]\, dz = -8P(\tau)^k \int_{\Delta_1 \cup \Delta_2} u \exp\left(-kdu^3\right) (1+O(k\delta^4))\, du\,.$$

The integral along $\Delta_1 \cup \Delta_2$ can be extended to two full half lines of angle $\pm 2\pi/3$ emanating from the origin, this at the expense of introducing only exponentially small error terms (since $n\delta^3 \to \infty$). The rescaling $v = u(kd)^{1/3} \exp(2i\pi/3)$ on Δ_1 and $v = u(kd)^{1/3} \exp(-2i\pi/3)$ on Δ_2 then shows that the completed integral equals

$$(kd)^{-2/3}(e^{4i\pi/3} - e^{-4i\pi/3}) \int_0^{+\infty} v \exp(-v^3) dv = -(kd)^{-2/3} \frac{i}{\sqrt{3}} \Gamma(2/3),$$

where the evaluation results from a cubic change of variable. In summary, we have found

$$[z^n]M^k(z) = \frac{1}{6i\pi} (\mathcal{E}_0 + \mathcal{E}_{1,2}) = \frac{3^{5/6}}{12\pi} \frac{P(\tau)^k}{k^{2/3}} \Gamma(2/3) \left(1 + O(k\delta^4)\right),$$

which, given our choice of δ, is equivalent to the statement.

A similar reasoning proves that the estimate remains valid for $n = 3k + e$ with $e = 1$ or $e = 2$, and more generally with any e satisfying $e = O(1)$.

Nearby saddles When k is close to $n/3$, we choose in the representation (9) an integration contour Γ that catches *simultaneously* the contributions of the two saddle points z_- and z_+. For this purpose, we adopt a contour that goes through the mid-point, $\zeta := (z_- + z_+)/2$, and, like in the previous case, meets the positive real line at an angle of $\pm 2\pi/3$. Local estimates of the integrand, once suitably normalized, lead to a complex integral representation that eventually reduces to Airy functions.

Theorem 3 (Local limit law and nearby saddles). *The probability distribution* $\Pr(X_n = k)$ *admits a local limit law of the map–Airy type: for any real numbers* a, b*, one has*

$$\sup_{a \le \frac{k-n/3}{n^{2/3}} \le b} \left| n^{2/3} \Pr(X_n = k) - \frac{16}{81} \frac{3^{4/3}}{4} \mathcal{A}\left(\frac{3^{4/3}}{4} \frac{k - n/3}{n^{2/3}} \right) \right| \to 0.$$

Proof. We set $k = n/3 + xn^{2/3}$ where x lies in a finite interval of the real line, and define $H := \ln(\psi^{k/n}\phi/z)$ (this replaces $\ln(P)$ in the previous argument). The starting point is again the integral representation (9) taken along a contour Γ that comprises Γ_0, a circle minus a small arc, together with two connecting small segments Δ_1, Δ_2 of length δ, now meeting at ζ, where δ is chosen according to the requirement (11). The arc Γ_0 lies below the level curve of ζ, and the corresponding contribution $\mathcal{E}_0$ is estimated to be exponentially negligible.

We turn next to the contribution $\mathcal{E}_{1,2}$ arising from $\Delta_1 \cup \Delta_2$. The distance between the two saddle points z_-, z_+ is $O(n^{-1/3})$ which represents the "scale" of the problem.

One thus sets $z = \zeta + vn^{-1/3}$. Local expansions of H and G are then best carried out with the help (suitably monitored!) of a computer algebra system like `Maple`. The computation relies on the assumption $x = O(1)$, but some care in performing expansions is required because of the relations (11). We find eventually

$$\mathcal{E}_{1,2} = \left(\frac{27}{4}\right)^n 4^{-k} n^{-2/3} \exp\left(-\frac{27}{32}x^3\right) \int_{\Delta_1' \cup \Delta_2'} (9x/2 - 8v) \exp\left(-\frac{64}{27}v^3 - \frac{9}{4}x^2 v\right)(1+\eta)\, dv,$$

where the error term η satisfies $\eta = O(\delta^4 n + n^{-1/12})$ and the segments Δ_1', Δ_2' each have length $\delta n^{1/3}$ tending to infinity according to our assumptions. Perform finally the change of variable $v = \left(\frac{64}{9}\right)^{-1/3} t$ and complete the integration path to $e^{\pm 2i\pi/3}\infty$: the integral then reduces to $\mathrm{Ai}(x), \mathrm{Ai}'(x)$ through contour integrals representations equivalent to (1) (by Cauchy's theorem, with integration path $e^{\pm 2i\pi/3}\infty$ changed to $e^{\pm i\pi/2}\infty$). Thus, for $x = O(1)$ and $k = n/3 + xn^{2/3}$, the main estimate found is

$$[z^n]M^k(z) = \frac{k}{n} 4^{-k} \left(\frac{27}{4}\right)^n n^{-2/3} \frac{3^{4/3}}{4} \mathcal{A}\left(\frac{3^{4/3}}{4}x\right)(1 + o(1)),$$

where $\mathcal{A}(x)$ is the map–Airy density function. This form is equivalent to the statement. The argument also gives a speed of convergence to the limit law of $O(n^{-1/12+o(1)})$.

3 Coalescing Saddles

In the present section, we provide a uniform description of the transition regions around $n/3$, allowing k to range anywhere $o(n)$ and $n - o(n)$, precisely, between $\lambda(n)$ and and $n - \lambda(n)$, for any $\lambda(n) = o(n)$ with $\lambda(n) \to \infty$. For the study of this wide region in the scale of n, we set

$$k = \alpha_0 n + \beta n = (1/3 + \beta)n,$$

with estimates valid uniformly for β in any compact subinterval of $]-\frac{1}{3}, \frac{2}{3}[$.

Theorem 4 (Large range and coalescent saddles). *Let $k = n(1/3 + \beta)$, and γ, a_1, a_4 be the functions of β given below. Let $\lambda(n)$ be any function with $\lambda(n) = o(n)$ and $\lambda(n) \to +\infty$. Then, with $\chi = n^{1/3}\gamma$, $\Pr(X_n = n/3 + \beta n)$ equals*

$$\frac{16}{81(1+3\beta)^{3/2} n^{2/3}} \left(\frac{a_1}{2}\mathcal{A}(\chi) + \frac{a_4}{n^{2/3}} \exp\left(-\frac{2}{3}\chi^3\right) \mathrm{Ai}(\chi^2)\right)(1 + O(1/n)), \tag{12}$$

where the error term is uniform for β in any compact subinterval of $]-\frac{1}{3}, \frac{2}{3}[$ and, up to replacing $O(1/n)$ by $O(\lambda(n)^{-1})$, it is also uniform for any $k > \lambda(n)$. With $\mathcal{L}(x) = x \ln x$, the quantities γ, a_1, and a_4 are:

$$\gamma = \left(2\mathcal{L}(1+3\beta/4) - \frac{1}{2}\mathcal{L}(1-3\beta/2) - \frac{1}{4}\mathcal{L}(1+3\beta) - \frac{9}{4}\beta \ln 2\right)^{1/3} \tag{13}$$

$$\frac{a_1}{2} = \frac{9}{8}\left(\frac{\beta/\gamma}{(1+3\beta/4)(1-3\beta/2)(1+3\beta)}\right)^{1/2} \qquad a_4 = \frac{4}{9\beta^2}\sqrt{\frac{\gamma}{\beta}} - \frac{a_1}{4\gamma^2} \tag{14}$$

The estimates involve Airy functions composed with the quantity χ that depends nonlinearly on β. In particular, formula (12) extends the estimates of Section 2 when $k = n/3 + xn^{2/3}$, since in that case $\chi \propto x$ while $\beta \to 0$ and the following approximations apply:

$$\gamma = \frac{3^{4/3}}{4}\beta + O(\beta^2), \quad \frac{a_1}{2} = \frac{3^{4/3}}{4} + O(\beta), \quad a_4 = -\frac{15}{64}3^{2/3} + O(\beta), \quad \beta \to 0.$$

(The resulting speed of convergence to the Airy law appears to be $O(n^{-2/3})$.) As soon as k leaves the $n/3 \pm O(n^{2/3})$ region, the two Airy terms in (12) start interfering and large deviations are then precisely quantified by (12). When k drifts away to the left of $n/3$ (and $\chi \to -\infty$), basic asymptotics of Airy functions show that the formula simplifies to agree with the results of Section 2.

Proof. The transition phenomenon to be described is the coalescence of two simple saddle points into a double one; see [3,24]. The simplest occurrence of the phenomenon appears in the integration of $\exp[nf(t,\gamma)]$ with

$$f'(t,\gamma) = t^2 - \gamma^2.$$

Indeed in this case there are two saddle points $\pm\gamma$, coalescing into a double saddle point as $\gamma \to 0$. The strategy consists in performing a change of variable in order to reduce the original problem (9) to this simpler case. Denote the kernel of the integral as $H(z,\beta) = \ln(\psi^{k/n}\phi/z)$ with $k = (1/3+\beta n)$ and the dependency on β made explicit. The integral in (9) is

$$I(n,\beta) = \int_\Gamma G(z)\exp[nH(z,\beta)]dz,$$

and we seek a change of variable of the form

$$H(z,\beta) = -\left(t^3/3 - \gamma^2 t\right) + r = f(t,\gamma). \tag{15}$$

It turns out that, taking $\gamma = \gamma(\beta)$ to be the real cubic root of $\gamma^3 = \frac{3}{4}[H(z_+,\beta) - H(z_-,\beta)]$, (the relation is expressed by (13)) and $r = r(\beta)$ to be

$$r = \frac{1}{2}[H(z_+,\beta) + H(z_-,\beta)] = H(z_+,\beta) - \frac{2}{3}\gamma^3 = \ln(\psi(\tau)^{k/n}/\rho) - \frac{2}{3}\gamma^3, \tag{16}$$

there exists a conformal map $z \to t$ from the disc D of diameter $[\frac{1}{4}, \frac{3}{4}]$ to a domain D_β satisfying (15) and mapping $z_\pm$ onto $\pm\gamma$. For simplicity, we restrict β to $[-\frac{1}{4}, \frac{1}{4}]$. The domain D_β contains the disc D' of diameter $[-\frac{1}{4}, \frac{1}{4}]$. Let us denote by $z(t)$ the inverse mapping and $G_0(t,\beta) = G(z(t))\dot{z}(t)$ where $\dot{z}(t) = \frac{dz}{dt}$. Remark that $G_0(t,\beta)$ is regular in D'. To guide his intuition, the reader may think of the map $z \to t$ as a slight deformation of the map $z \to 2(z-r)$.

Let us now proceed with the integral. As is usual with saddle point integrals we first localise the integral in D, neglecting the parts of the path down in valleys,

$$I(n,\beta) = \int_\Gamma G(z)\exp[nH(z,\beta)]\,dz = \int_{\Gamma\cap D} G(z)\exp[nH(z,\beta)]\,dz + \mathcal{E}_1(n,\beta),$$

where $\mathcal{E}_1(n,\beta)$ is exponentially negligible when $n \to \infty$, uniformly in β. Inside the disc D we apply the change of variables (15), then restrict attention to the disc D', and

deform the contour onto the relevant part of $\Delta_\infty = \{te^{\pm\frac{2i\pi}{3}},\, t \geq 0\}$:

$$\begin{aligned} I(n,\beta) &= \int_{\Gamma_\beta \cap D_\beta} G(z(t)) \exp[nf(t,\gamma)]\dot{z}(t)\, dt + \mathcal{E}_1(n,\beta) \\ &= \int_{\Delta_\infty \cap D'} G_0(t,\beta) \exp[nf(t,\gamma)]\, dt + \mathcal{E}_2(n,\beta). \end{aligned}$$

In order to evaluate this integral one needs to dispose of the modulation factor $G_0(t,\beta)$. This can be done via an integration by part: A local expansion near γ yields

$$G_0(t,\beta) = (\gamma - t)a_1 + (t^2 - \gamma^2)H_0(t,\beta),$$

where $H_0(t,\beta)$ is regular in D', and a_1 is given by (14). The integral $I(n,\beta)$ is thus

$$I(n,\beta) = \exp(nr) \int_{\Delta_\infty \cap D'} (\gamma - t)a_1 \exp\left(-n\left(t^3/3 - \gamma^2 t\right)\right)\, dt + R_0(n,\beta),$$

where after integration by part, and up to another exponentially negligible term,

$$R_0(n,\beta) = \frac{\exp(nr)}{n} \int_{\Delta_\infty \cap D'} \left(\frac{d}{dt}H_0(t,\beta)\right) \exp\left[-n\left(\frac{t^3}{3} - \gamma^2 t\right)\right] dt + \mathcal{E}_3(n,\beta).$$

The integration by part has reduced the order of magnitude by a factor n, but $R_0(n,\beta)$ is amenable to the same treatment as $I(n,\beta)$. We shall content ourselves with the next terms: let $\frac{d}{dt}H_0(t,\beta) = a_2\gamma + a_3 t + (t^2 - \gamma^2)H_1(t,\beta)$, with $H_1(t,\beta)$ regular in D', a_2, a_3 functions of β, so that we have

$$I(n,\beta) = \exp(nr) \int_{\Delta_\infty} \left(\gamma\left(a_1 + \frac{a_2}{n}\right) - t\left(a_1 - \frac{a_3}{n}\right)\right) \exp\left[-n\left(\frac{t^3}{3} - \gamma^2 t\right)\right] dt + R_1(n,\beta).$$

where the integral has been extended to the whole of Δ_∞ at the expense of yet another exponentially negligible term. The error term is

$$R_1(n,\beta) = \frac{\exp(nr)}{n^2} \int_{\Delta_\infty \cap D'} \left(\frac{d}{dt}H_1(t,\beta)\right) \exp\left[-n\left(\frac{t^3}{3} - \gamma^2 t\right)\right] dt + \mathcal{E}_4(n,\beta).$$

In terms of the Airy function, we thus have

$$I(n,\beta) = 2i\pi\frac{\exp(nr)}{n^{2/3}}\left(\gamma n^{1/3}\left(a_1 + \frac{a_2}{n}\right)\mathrm{Ai}(n^{2/3}\gamma^2) - \left(a_1 - \frac{a_3}{n}\right)\mathrm{Ai}'(n^{2/3}\gamma^2)\right) + R_1(n,\beta),$$

and the error term $R_1(n,\beta)$ can be estimated: there exist d_0 and d_1 positive such that

$$|R_1(n,\beta)| \leq \frac{\exp(nr)}{n^2}\left(\frac{d_0}{n^{1/3}}|\mathrm{Ai}(n^{2/3}\gamma^2)| + \frac{d_1}{n^{2/3}}|\mathrm{Ai}'(n^{2/3}\gamma^2)|\right).$$

The theorem follows from formulae (6), (8), (16) and the definition of the map–Airy law, upon setting $a_4 = \gamma(a_2 + a_3)$.

4 Applications to Maps and Random Sampling

The results obtained in the particular case of 3-connected cores of nonseparable maps are instances of a very general pattern in the physics of random maps. Indeed all families in the table below obey the Lagrangean framework and are amenable to the saddle point methods developed in previous sections.

Table 1. A selection of composition schemes ($\mathcal{X}$ an edge, $\mathcal{L}, \mathcal{D}$ auxiliary families).

maps ($\mathcal{M}$), $\mathcal{M}_n$	cores ($\mathcal{C}$), scheme	α_0	c
general, n edges	nonseparable, $\mathcal{M} \simeq \mathcal{C}[\mathcal{X}\mathcal{M}^2]$	1/3	$3/4^{2/3}$
general, n edges	bridgeless, $\mathcal{M} \simeq \mathcal{C}[\mathcal{X}(\mathcal{X}\mathcal{M})^*]$	4/5	$(5/3)^{2/3}/4$
general, n edges	loopless, $\mathcal{M} \simeq \mathcal{L} + \mathcal{C}[\mathcal{X}((\mathcal{X}\mathcal{M})^*)^2]$	2/3	3/2
loopless , n edges	simple, $\mathcal{M} \simeq \mathcal{C}[\mathcal{X}\mathcal{M}]$	2/3	$3^{4/3}/4$
bipartite, n edges	bip. simples, $\mathcal{M} \simeq \mathcal{C}[\mathcal{X}\mathcal{M}]$	5/9	$3^{8/3}/20$
bipartite, n edges	bip. nonsep., $\mathcal{M} \simeq \mathcal{C}[\mathcal{X}\mathcal{M}^2]$	5/13	$(13/6)^{5/3} \cdot 3/10$
bipartite, n edges	bip. bridgeless, $\mathcal{M} \simeq \mathcal{C}[\mathcal{X}(\mathcal{X}\mathcal{M})^*]$	3/5	$(15/2)^{5/3}/18$
nonsep., n edges	simple nonsep., $\mathcal{M} \simeq \mathcal{C}[\mathcal{X}\mathcal{M}]$	4/5	$15^{5/3}/36$
nonsep., $n+1$ edges	3-connected, $\mathcal{M} \simeq \mathcal{D} + \mathcal{C}[\mathcal{M}]$	1/3	$3^{4/3}/4$
cubic nonsep., $n+2$ faces	cubic 3-conn., $\mathcal{M} \simeq \mathcal{C}[\mathcal{X}(1+\mathcal{M})^3]$	1/2	$(3/2)^{1/3}$
cubic 3-conn., $n+2$ faces	cubic 4-conn., $\mathcal{M} \simeq \mathcal{M} \cdot \mathcal{C}[\mathcal{X}\mathcal{M}^2]$	1/2	$6^{2/3}/3$

Theorem 5. *Consider any scheme of Table 1 with parameters α_0 and c. The probability $\Pr(X_n = k)$ that a map of size n has a core of size k has a local limit law of the map–Airy type with centering constant α_0 and scale parameter c.*

The technique of [11] relates the size of the core to the size of the largest component in random maps. Also, since maps have almost surely no symmetries [17], the analysis extends to unrooted maps. As a consequence:

Theorem 6. (i) *Consider any scheme of Table 1 with parameters α_0 and c. Let X_n^* be the size of the largest component of in a random map of size n with uniform distribution. Then*

$$\Pr\left(X_n^* = \lfloor \alpha_0 n + x n^{2/3} \rfloor\right) = \frac{c\mathcal{A}(cx)}{n^{2/3}}\,(1 + O(n^{-2/3})),$$

uniformly for x in any bounded interval. Furthermore, if x is restricted to the shorter range $|x| < \lambda(n)^{-1}$ for a fixed function $\lambda(n)$ going to infinity with n, then

$$\Pr\left(X_n^* = \lfloor \alpha_0 n + x n^{2/3} \rfloor\right) = \frac{c}{n^{2/3}} \frac{3^{1/6}\Gamma(2/3)}{\pi}\,(1 + O(\lambda(n)^{-1})).$$

(ii) *The same results hold for random unrooted maps.*

Theorem 6 extends results of Bender, Gao, Richmond, and Wormald [2,11] who proved that X_n^* lies in the range $\alpha_0 n \pm \lambda(n) n^{2/3}$ with probability tending to 1, where $\lambda(n)$ is any function going to infinity with n.

Random sampling algorithms for various families of planar maps were described in [19]. For general, nonseparable, bipartite, and cubic nonseparable maps, an algorithm `Map` is given there that takes an integer n and outputs in linear time a map of size n uniformly at random. For the other families of Table 1, a probabilistic algorithm `Core` described below is used.

Probabilistic algorithm `Core(k)` with parameter $f(k)$
1. use `Map(n)` to generate a random map $M \in \mathcal{M}$ of size $n = f(k)$;
2. extract the largest component C of M with respect to the scheme;
3. if C does not have size k, then go back to step 1;
4. output C.

Safe for a set of measure that is exponentially small, this algorithm produces a uniform element of $\mathcal{C}_k$. The expected number of loops made by `Core` is exactly $\ell_n = \Pr(X_n = k)^{-1}$. The results of the paper enable us to precisely analyse this and a number of related algorithms of [18,19]. We cite just here:

Theorem 7. *In all extraction/rejection algorithms of [19], the choice $f(k) = n/\alpha_0$ yields an algorithm whose average number of iterations satisfies*

$$\ell_n \sim n^{2/3}/(\mathcal{A}(0)c).$$

Let $x_0 \approx 0.44322$ be the position of the peak of the map-Airy density function $((1 - 4x_0^3)\mathrm{Ai}(x_0^2) + 4x_0^2\mathrm{Ai}'(x_0^2) = 0)$. The optimal choice $f(k) = k/\alpha_0 - \frac{x_0}{\alpha_0 c}(k/\alpha_0)^{2/3}$ reduces the expected number of loops by $1 - \mathcal{A}(0)/\mathcal{A}(x_0) \approx 30\%$.

This proves that the extraction/rejection algorithms have overall complexity $O(k^{5/3})$, as do variant algorithms of [18,19] that are uniform over all $\mathcal{C}_k$. The complexity becomes $O(k)$ if some small tolerance is allowed on the size of the multiply connected map generated. Theorems of the paper enable us to quantify precisely various trade-offs and fine-tune algorithms (details in the full paper). As exemplified by Fig. 1(ii), the predictions fit nicely with experimental results.

5 Composition of Singularities

Map enumeration can be approached through the Lagrangean framework and the saddle point analysis developed so far takes off from there. An alternative approach to the problem relies on singularity analysis [9], as introduced in Section 1. The results of this section contribute to the general classification of combinatorial schemas according to the nature of their singularities [20].

First, a definition. Let M and C be two generating functions with dominant singularities at ρ and σ, such that $M(z) = \sigma - a(1-z/\rho) + b(1-z/\rho)^{3/2} + O((1-z/\rho)^2)$, and $C(z) = c_0 - a'(1-z/\sigma) + b(1-z/\sigma)^{3/2} + O((1-z/\sigma)^2)$, in an indented domain extending beyond the circle of convergence (see [9]). Then the bivariate substitution scheme $C(uM(z))$ is said to be a *critical composition scheme* of type $(3/2, 3/2)$. The functional composition $C(uM(z))$ describes the size of the $\mathcal{C}$ component in a combinatorial substitution $\mathcal{C}[\mathcal{M}]$. The scheme is called critical since the singular value of the inner function (M) equals the singularity of the outer function (C). It will be recognized that Tutte's construction is an instance (with σ replacing the map specific $\psi(\tau)$ of formulae (5) and (7)). Schemes of this broad form have been only scantily analysed, a notable exception being the critical composition scheme of type $(-1, 3/2)$ that shows up in ordered forests and in random mappings (functional graphs): in that case, the density is known

to be of the Rayleigh type [6,20]. The results of this section somehow recycle in a different realm the intuition gathered by the method of coalescing saddles, although the technical developments are a bit different.

Theorem 8. *(i) For $k = \alpha n + \lambda(n)$, with $0 \le \alpha < \alpha_0 = \frac{\sigma}{a}$ and $\lambda(n) = o(n)$, the probability distribution of the size X_n of the $\mathcal{C}$–component of random element of $\mathcal{C}[\mathcal{M}]$ of size n satisfies*

$$\Pr(X_n = k) \sim \tfrac{b'}{\sigma\sqrt{\pi}}\, \alpha^{-3/2}(1-\alpha/\alpha_0)^{-5/2}\, n^{-3/2} \qquad \text{for } \alpha > 0;$$
$$\Pr(X_n = k) \sim \tfrac{b'}{\sigma\sqrt{\pi}}\, \lambda(n)^{-3/2} \qquad \text{for } \alpha = 0 \text{ and } \lambda(n) \to +\infty\,.$$

(ii) For $k = \alpha_0 n + x n^{2/3}$, $\alpha_0 = \sigma/a$, $x = o(n^{1/3})$, an Airy-map law holds:

$$n^{2/3}\Pr(X_n = \alpha_0 n + x n^{2/3}) \;\sim\; \tfrac{b'}{\alpha_0^{3/2} b} c\mathcal{A}(cx) \qquad \textit{where } c = \left(\tfrac{b}{3a}\right)^{2/3}/\alpha_0.$$

The proof relies on a modification of the Hankel contour used in classical singularity analysis together with a different scaling. It will be developed in the full paper. The theorem is a companion to Theorems 1, 2, 3, 4 that can also be used to analyse forests of unrooted trees [14] in the critical region, a problem itself relevant to the emergence of the giant component in random graphs [13,14].

References

1. Carl M. Bender and Steven A. Orszag. *Advanced mathematical methods for scientists and engineers. I.* Springer-Verlag, New York, 1999. Asymptotic methods and perturbation theory, Reprint of the 1978 original.
2. Edward A. Bender, L. Bruce Richmond, and Nicholas C. Wormald. Largest 4-connected components of 3-connected planar triangulations. *Random Structures & Algorithms*, 7(4):273–285, 1995.
3. Norman Bleistein and Richard A. Handelsman. *Asymptotic Expansions of Integrals.* Dover, New York, 1986. A reprint of the second Holt, Rinehart and Winston edition, 1975.
4. Béla Bollobás, Christian Borgs, Jennifer T. Chayes, Jeong Han Kim, and David B. Wilson. The scaling window of the 2-sat transition. Preprint, 1999. Available as document math.CO/9909031 at the Los Alamos archive `http://xxx.lanl.gov/form/`.
5. N. G. De Bruijn. *Asymptotic Methods in Analysis.* Dover, 1981. A reprint of the third North Holland edition, 1970 (first edition, 1958).
6. Michael Drmota and Michèle Soria. Marking in combinatorial constructions: Generating functions and limiting distributions. *Theoretical Computer Science*, 144(1–2):67–99, June 1995.
7. Philippe Duchon. Q-grammars and wall polyominoes. *Annals of Combinatorics*, 3:311–321, 1999.
8. P. Flajolet, D. E. Knuth, and B. Pittel. The first cycles in an evolving graph. *Discrete Mathematics*, 75:167–215, 1989.
9. Philippe Flajolet and Andrew M. Odlyzko. Singularity analysis of generating functions. *SIAM Journal on Applied Mathematics*, 3(2):216–240, 1990.
10. Philippe Flajolet, Patricio Poblete, and Alfredo Viola. On the analysis of linear probing hashing. *Algorithmica*, 22(4):490–515, December 1998.

11. Zhicheng Gao and Nicholas C. Wormald. The size of the largest components in random planar maps. *SIAM J. Discrete Math.*, 12(2):217–228 (electronic), 1999.
12. Ian P. Goulden and David M. Jackson. *Combinatorial Enumeration.* John Wiley, New York, 1983.
13. Svante Janson, Donald E. Knuth, Tomasz Łuczak, and Boris Pittel. The birth of the giant component. *Random Structures & Algorithms*, 4(3):233–358, 1993.
14. V. F. Kolchin. *Random Graphs*, volume 53 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, Cambridge, U.K., 1999.
15. G. Louchard. The Brownian excursion: a numerical analysis. *Computers and Mathematics with Applications*, 10(6):413–417, 1984.
16. Thomas Prellberg. Uniform q-series asymptotics for staircase polygons. *Journal of Physics A: Math. Gen.*, 28:1289–1304, 1995.
17. L. B. Richmond and N. C. Wormald. Almost all maps are asymmetric. *J. Combin. Theory Ser. B*, 63(1):1–7, 1995.
18. Gilles Schaeffer. *Conjugaison d'arbres et cartes combinatoires aléatoires.* PhD thesis, Université Bordeaux I, 1998.
19. Gilles Schaeffer. Random sampling of large planar maps and convex polyhedra. In *Proceedings of the thirty-first annual ACM symposium on theory of computing (STOC'99)*, pages 760–769, Atlanta, Georgia, may 1999. ACM press.
20. Michèle Soria-Cousineau. *Méthodes d'analyse pour les constructions combinatoires et les algorithmes.* Doctorate in sciences, Université de Paris–Sud, Orsay, July 1990.
21. Lajos Takacs. A Bernoulli excursion and its various applications. *Advances in Applied Probability*, 23:557–585, 1991.
22. W. T. Tutte. Planar enumeration. In *Graph theory and combinatorics (Cambridge, 1983)*, pages 315–319. Academic Press, London, 1984.
23. G. N. Watson. *A Treatise on the Theory of Bessel Functions.* Cambridge University Press, 1980.
24. Roderick Wong. *Asymptotic Approximations of Integrals.* Academic Press, 1989.

Analysing Input/Output-Capabilities of Mobile Processes with a Generic Type System*

Barbara König (koenigb@in.tum.de)

Fakultät für Informatik, Technische Universität München

Abstract. We introduce a generic type system (based on Milner's sort system) for the synchronous polyadic π-calculus, allowing us to mechanise the analysis of input/output capabilities of mobile processes. The parameter of the generic type system is a lattice-ordered monoid, the elements of which are used to describe the capabilities of channels with respect to their input/output-capabilities. The type system can be instantiated in order to check process properties such as upper and lower bounds on the number of active channels, confluence and absence of blocked processes.

1 Introduction

For the analysis and verification of processes there are basically two approaches: methods that are complete, but cannot be fully mechanised, and fully automatic methods which are consequently not complete, i.e. not all processes satisfying the property to be checked are recognised.

One promising direction for the latter approach is to use type or sort systems and type inference with rather complex types abstracting from process behaviour. In the last few years there have been several papers presenting such type systems for the polyadic π-calculus and other process calculi, checking e.g. input/output behaviour [15], absence of deadlocks [7], security properties [1, 4], allocation of permissions to names [16] and many others. Types are compositional and thus allow reuse of information obtained in the analysis of smaller subsystems.

One drawback of the type systems mentioned above is the fact that they are specialised to check very specific properties. A much more general approach is a theory of types by Honda [6] which is based on typed algebras and gives a classification of type systems. This theory is very general and it is thus necessary to prove the subject reduction property and the correctness of a type system for every instance. Our paper attempts to fill the gap between the two extremes. We present a generic type system where we can show the subject reduction property for the general case, and by instantiating the type system we are able to analyse specific properties of processes. Despite its generality, our type system can be used to generate existing type systems, or at least subsets of them. With the introduction of residuation (explained below) we can even type some processes which are not typable by comparable type systems.

We concentrate on properties connected to input/output capabilities of processes in the synchronous polyadic π-calculus. In our examples (see section 5) we check properties such as upper and lower bounds on the number of active channels, confluence,

* Research supported by SFB 342 (subproject A3) of the DFG.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 403-414, 2000
© Springer-Verlag Berlin Heidelberg 2000

absence of blocked input or output prefixes. Determining these capabilities of a process involves counting and we attempt to keep this concept as general as possible by basing the generic type system on commutative monoids. Instantiating a type system mainly involves choosing an appropriate monoid, and monoid elements associated with input and output prefixes (e.g. for counting the number of prefixes with a certain subject).

Instead of giving the precise answer to every question, our type system uses over-approximation (e.g. we can expect results of the form "there are at most two active channels with subject x at any given time"). Hence plain monoids are not sufficient, but we need ordered monoids (so-called lattice-ordered monoids or l-monoids), equipped with a partial order compatible with summation.

There is a huge class of lattice-ordered monoids which are residuated, i.e. some limited form of subtraction can be defined. Residuation can be put to good use in process analysis. Consider, e.g. the process $P = \bar{x}.x.\mathbf{0}$. While P increases the number of occurrences of the output prefix $\bar{x}$ by one, it does not do so for the input prefix x, since we are interested exclusively in the number of prefixes on the outer level (i.e. in prefixes which are currently active) and x can only be reached by a communication with $\bar{x}$ which decreases the number of input prefixes in the environment by one. This decrease can be anticipated when typing P, and is taken into consideration by subtracting one from the number of input prefixes.

The type of a process contains an assignment of names to sorts and a mapping of sorts to strings of sorts (as in [13]), keeping track of channel arities, i.e. if channel x has sort s, and n-ary tuples are communicated via x, then s will be mapped to a string of sorts having length n, being the sorts of the respective channels. Thus, successful typing also guarantees the absence of runtime errors produced by mismatching arities. Furthermore a monoid element is assigned to each sort s. The monoid element is expected to be an upper bound for the capabilities of all channels having sort s.

2 Preliminaries

2.1 The π-Calculus

The π-calculus [12, 13] is an influential paradigm describing communication and mobility of processes. In this paper we will consider the synchronous polyadic π-calculus without choice and matching, and replication is only defined for input prefixes. Its syntax is as follows:

$$P ::= \mathbf{0} \mid (\nu x : s)P \mid P_1|P_2 \mid \bar{x}\langle\tilde{z}\rangle.P \mid x(\tilde{y}).P \mid !x(\tilde{y}).P$$

where s is an element from a fixed set of sorts S and x is taken from a fixed set of names $\mathcal{N}$. $\tilde{y} = y_1 \ldots y_n$ and $\tilde{z} = z_1 \ldots z_n$ are abbreviations for sequences with elements from $\mathcal{N}$. We call $\bar{x}\langle\tilde{z}\rangle$ *output prefix* and $x(\tilde{y})$ *input prefix*.

The set of all free names (i.e. names not bound by either ν or by an input prefix) of a process P is denoted by $fn(P)$. The process obtained by replacing the free names y_i by x_i in P (and avoiding capture) is called $P\{\tilde{x}/\tilde{y}\}$.

Structural congruence is the smallest congruence obeying the rules in the upper part of table 1, and equating processes that can be converted into one another by consistent

renaming of bound names (α-conversion). We use a reduction semantics as for the chemical abstract machine [2] instead of a labelled transition semantics.

Structural Congruence	(C-COM) $P_1 \mid P_2 \equiv P_2 \mid P_1$ (C-**0**) $P \mid \mathbf{0} \equiv P$
	(C-ASS) $P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3$
	(C-RESTR1) $(\nu x\colon s)(\nu y\colon t)P \equiv (\nu y\colon t)(\nu x\colon s)P$ if $x \neq y$
	(C-RESTR2) $((\nu x\colon s)P_1) \mid P_2 \equiv (\nu x\colon s)(P_1 \mid P_2)$ if $x \notin fn(P_2)$
Reduction Rules	(R-COMM) $\bar{x}\langle\tilde{z}\rangle.Q \mid x(\tilde{y}).P \rightarrow Q \mid P\{\tilde{z}/\tilde{y}\}$
	(R-REP) $\bar{x}\langle\tilde{z}\rangle.Q \mid !x(\tilde{y}).P \rightarrow Q \mid P\{\tilde{z}/\tilde{y}\} \mid !x(\tilde{y}).P$
	(R-PAR) $\dfrac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q}$ (R-RESTR) $\dfrac{P \rightarrow P'}{(\nu x\colon s)P \rightarrow (\nu x\colon s)P'}$
	(R-EQU) $\dfrac{Q \equiv P, P \rightarrow P', P' \equiv Q'}{Q \rightarrow Q'}$

Table 1. operational semantics of the π-calculus

Consider the following processes which we will use as an example in this paper (we omit the final **0**):

$$F = c(r).\bar{d}\langle r\rangle.d(a).\bar{c}\langle a\rangle \quad S = d(s).s(h_1, h_2).\bar{d}\langle h_1\rangle \quad T = \bar{c}\langle h\rangle.c(x) \quad H = \bar{h}\langle i_1, i_2\rangle$$

There is a forwarder F which receives requests on a channel c, forwards them on a channel d to a server, receives the answer and sends it back on c. The server S receives requests on d, and we assume that these requests come with a name s where the server can get further information. The server obtains this information, processes it and sends the answer back on d (in our example we keep the "processing part" very simple, the server just sends back the first component). Furthermore T is a trigger process, starting the execution of F and receiving the result in the end, and H delivers information to the server.

We can combine the processes F, S, T, H to obtain P as the entire system. If we want F and S to be persistent, we regard P'.

$$P = T \mid H \mid (\nu d\colon s_d)(F \mid S) \qquad P' = T \mid H \mid (\nu d\colon s_d)(!F \mid !S)$$

A programmer analysing this piece of code might be interested in the following properties: input/output behaviour, upper and lower bound on the number of channels being active, confluence properties and absence of blocked prefixes that never find a communication partner. E.g., examining P will reveal that at any given time every name is used for input and output at most once and that P is therefore confluent.

2.2 Residuated Lattice-Ordered Monoids

Lattice-ordered monoids are a well-developed mathematical concept (see e.g. [3]). We are interested in commutative residuated l-monoids in order to represent input/output capabilities.

Definition 1. (Lattice-ordered Monoid)
A commutative lattice-ordered monoid (l-monoid) *is a tuple* $(I, +, \leq)$ *where* I *is a set,* $+ : I \times I \to I$ *is a binary operation and* $\leq$ *is a partial order which satisfy:*

- $(I, +)$ *is a commutative monoid, i.e.* $+$ *is associative and commutative, and there is a unit* 0 *with* $0 + a = a$ *for every monoid element* $a \in I$.
- $(I, \leq)$ *is a lattice, i.e.* $\leq$ *is a partial order, where two elements* $a, b \in I$ *have a* join *(or least upper bound)* $a \vee b$ *and a* meet *(or greatest lower bound)* $a \wedge b$.
- I *contains a bottom element* $\bot$*, the smallest element in* I*, and a top element* $\top$*, the greatest element in* I.
- *For* $a, b, c \in I$*:* $a + (b \vee c) = (a + b) \vee (a + c)$ *and* $a + (b \wedge c) = (a + b) \wedge (a + c)$

Any l-monoid $(I, +, \leq)$ is associated with an l-monoid $(I, \oplus, \leq)$ where $a \oplus b = (a + b) \vee a \vee b$ and $\bot$ is the unit. The significance of $\oplus$ can be made clear with the following consideration: monoid elements will be used to label sorts, being an upper bound for the capabilities of channels having this sort. E.g., we assume that a free name x and a bound name y have sort s, indicating that, during reduction, x might replace y. The capabilities of x and y are a respectively b. What capability should be associated with s? In the presence of positive monoid elements only, $a + b$ is the correct answer. If, however, a is negative, $a + b$ is actually smaller than b and if x has not yet replaced y, the monoid element associated with s underestimates the capabilities of y. Since we use over-approximation the correct sort label is $a \oplus b$.

Definition 2. (Residuated l-monoid) *Let* $(I, +, \leq)$ *be an l-monoid and let* $a, b \in I$. *The* residual $a - b$ *is the smallest* x *(if it exists) such that* $a \leq x + b$. I *is called* residuated *if all residuals* $a - b$ *exist in* I *for* $a, b \in I$.

Example: one residuated l-monoid which we will later use for the analysis of processes is $IO = (\{none, I, O, both\}, \vee, \leq)$ where $none \leq I \leq both$, $none \leq O \leq both$ and the monoid operation is the join, i.e. the l-monoid degenerates to a lattice. A channel name has for example capability O if it is used at most for output and capability *both* if it may be used for both output and input.

In order to count the number of inputs or outputs we use the residuated l-monoid $\mathbb{Z}^\infty = (\mathbb{Z} \cup \{\infty, -\infty\}, +, \leq)$ with all integers including ∞ and $-\infty$ ($\infty + (-\infty) = -\infty$). Residuation is subtraction for all monoid elements different from ∞ and $-\infty$.

The cartesian product of two l-monoids, e.g. $\mathbb{Z}^\infty \times \mathbb{Z}^\infty$, is also an l-monoid.

We use the following inequations concerning residuated l-monoids: for all elements a, b, c of a residuated l-monoid it holds that

$$a \leq (a - b) + b \qquad (a + b) - b \leq a \qquad (a + b) - c \leq (a - c) + b$$
$$(a + b) \vee 0 \leq (a \vee 0) + (b \vee 0) \qquad a + b \leq a \oplus b \qquad \bot + \bot = \bot \qquad \top + \top = \top$$

And we define: $sig(a) = \begin{cases} \bot \text{ if } a < 0 & 0 \quad \text{if } a = 0 \\ \top \text{ if } a > 0 & \textit{undefined} \text{ otherwise} \end{cases}$

3 The Type System and Its Properties

We define the notion of types and type assignments which have already been informally introduced in section 1.

Definition 3. (Type Assignment) *Let S be a fixed set of sorts and let $(I, +, \leq)$ be a fixed l-monoid. A type assignment $\Gamma = ob_\Gamma; m_\Gamma; x_1 \colon \langle s_1/a_1 \rangle, \ldots, x_n \colon \langle s_n/a_n \rangle$ (abbreviated by $ob_\Gamma; m_\Gamma; \tilde{x} \colon \langle \tilde{s}/\tilde{a} \rangle$) consists of a* sort mapping *$ob_\Gamma : S \to S^*$ (mapping sorts to* object sorts*), a mapping $m_\Gamma : S \to I$ (assigning a monoid element to every sort) and an assignment of channel names x_i to tuples consisting of a sort s_i and a monoid element a_i.*

We define $sort_\Gamma(x_i) = s_i$ and $\Gamma, y \colon \langle t/b \rangle$ denotes $ob_\Gamma; m_\Gamma; \tilde{x} \colon \langle \tilde{s}/\tilde{a} \rangle, y \colon \langle t/b \rangle$.

Sorts are used to control the mobility of names. That is if $ob_\Gamma(s) = s_1 \ldots s_n$, we know that only n-tuples of channel names with sorts s_i are sent or received via a channel with sort s. If a free name x and a bound name y have the same sort, we have to take into account that x may replace y during the reduction. We also use sorts as an intermediate level between names and monoid elements, since with α-conversion it is problematic to assign monoid elements directly to names.

Monoid elements appear in two places: in the range of m_Γ and in the tuples $x : \langle s/a \rangle$. The idea is to sum up the capabilities of x with $+$ in a while x is still free and add a to $m_\Gamma(s)$ with $\oplus$ as soon as x is hidden. We have to use $\oplus$ according to the explanation given in section 2.2. The other possibility would be to immediately add the capabilities to $m_\Gamma(s)$ with $\oplus$ (without storing them in a first), but since $a + b \leq a \oplus b$, this would lead to looser bounds. (It would, however, be possible in the case where we only consider monoid elements greater than or equal to 0, since in this case $+$ and $\oplus$ always coincide.)

In the rest of this paper we use the operations on type assignments given in table 2 (all operations on sequences are conducted pointwise): in (ADD-MON) we add a monoid element a to a type assignment Γ by adding a to $m_\Gamma(s)$ (with $\oplus$) and leaving everything else unchanged. And $\Gamma^{\langle \tilde{s}, \tilde{a} \rangle}$ denotes $(\ldots(\Gamma^{\langle s_1, a_1 \rangle})^{\langle s_2, a_2 \rangle} \ldots)^{\langle s_n, a_n \rangle}$.

$$\text{(ADD-MON)} \quad (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a} \rangle)^{\langle s, a \rangle} = ob; m'; \tilde{x} \colon \langle \tilde{s}/\tilde{a} \rangle$$

$$\text{where } m'(s') = \begin{cases} m(s) \oplus a & \text{if } s = s' \\ m(s) & \text{otherwise} \end{cases}$$

$$\text{(SUM)} \quad (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a} \rangle) \oplus (ob; m'; \tilde{x} \colon \langle \tilde{s}/\tilde{b} \rangle) = ob; m \oplus m'; \tilde{x} \colon \langle \tilde{s}/\tilde{a} + \tilde{b} \rangle$$

$$\text{(JOIN)} \quad (ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a} \rangle) \vee b = ob; m; \tilde{x} \colon \langle \tilde{s}/\tilde{a} \vee b \rangle$$

$$\text{(ORD)} \quad (ob; m; \tilde{x} : \langle \tilde{s}/\tilde{a} \rangle) \leq (ob; m'; \tilde{x} : \langle \tilde{s}/\tilde{b} \rangle) \iff m \leq m' \text{ and } \tilde{a} \leq \tilde{b}$$

$$\text{(REMOVE)} \quad \text{If } \Gamma = \Delta, x : \langle s/a \rangle, \text{ then } \Gamma \backslash \{x\} = \Delta$$

Table 2. Operations on type assignments

Summation $\Gamma \circledast \Gamma'$ (SUM) is defined for type assignments which contain the same names (having identical sorts) and which satisfy $ob_\Gamma = ob_{\Gamma'}$ (i.e. they have the same sort structure). In this case $m \oplus m'$ and $\tilde{a} + \tilde{b}$ denote the pointwise summation. The summation on type assignments has a counterpart (denoted by the same symbol) in Honda's work [6].

In (JOIN) a pointwise join with every monoid element assigned to a channel name and the monoid element b is defined. And finally we need a partial order on type assignments (ORD) and an operation removing an assumption on a name x from a type assignment (REMOVE).

We are now ready to define the rules of the type system (see table 3). *out* and *in* are fixed monoid elements (where *in* must be comparable to 0) representing the capabilities of output respectively input prefixes.

$$\frac{\Gamma \vdash P,\ \Gamma \leq \Delta}{\Delta \vdash P}\ (\text{T-}\leq) \qquad \Gamma \vee 0 \vdash \mathbf{0}\ (\text{T-NIL}) \qquad \frac{\Gamma_1 \vdash P_1,\ \Gamma_2 \vdash P_2}{\Gamma_1 \circledast \Gamma_2 \vdash P_1 \mid P_2}\ (\text{T-PAR})$$

$$\frac{\Gamma, x\colon \langle s/a\rangle, \tilde{z}\colon \langle \tilde{t}, \tilde{b}\rangle \vdash P}{(\Gamma, \tilde{z}\colon \langle \tilde{t}, \tilde{b}\rangle) \vee 0, x\colon \langle s/(a - \mathit{in}) \vee 0 + \mathit{out}\rangle \vdash \bar{x}\langle \tilde{z}\rangle.P}\ (\text{T-OUT}) \quad \text{if } ob_\Gamma(s) = \tilde{t}$$

$$\frac{\Gamma, x\colon \langle s/a\rangle, \tilde{y}\colon \langle \tilde{t}/\tilde{b}\rangle \vdash P}{(\Gamma \vee 0, x\colon \langle s/(a - \mathit{out}) \vee 0 + \mathit{in}\rangle)^{\langle \tilde{t}, \tilde{b}\rangle} \vdash x(\tilde{y}).P}\ (\text{T-IN}) \quad \text{if } ob_\Gamma(s) = \tilde{t}$$

$$\frac{\Gamma\backslash\{x\}, x\colon \langle s/a\rangle \vdash P}{\Gamma^{\langle x, a\rangle} \vdash (\nu x\colon s)P}\ (\text{T-RESTR}) \qquad \frac{\Gamma \vdash x(\tilde{y}).P}{\Delta, x\colon \langle s/a + \mathit{sig}(\mathit{in})\rangle \vdash !x(\tilde{y}).P}\ (\text{T-REP})$$

$$\text{if } \Gamma \circledast \Gamma \leq \Gamma \text{ and } \Gamma = \Delta, x\colon \langle t/a\rangle$$

Table 3. Rules of the type system

The intuitive meaning of the rules is as follows:

(T-$\leq$) We can always over-approximate the capabilities of a process.

(T-NIL) The nil process can have an arbitrary type assignment, provided the monoid elements of the free names are greater than 0.

(T-PAR) The parallel composition of two processes can be typed by adding their respective type assignments.

(T-OUT) First we subtract *in* from the monoid element a associated with the subject x of the output prefix, since the emergence of P means the removal of an input prefix with subject x somewhere else in the environment. We then take the join of all monoid elements and 0, since we only consider capabilities on the outer level of processes and thus we only consider future influence by positive capabilities, but not by negative ones (since we are doing over-approximation). In the end *out* is added to the monoid element associated with x.

Furthermore we have to check that the sort structure is correct, i.e. since $z_1, \ldots, z_n$ are communicated via x, the string of their sorts must be the object sort of the sort of x.

(T-IN) As described for (T-OUT), we subtract *out*, take the join with 0 and then add *in*. Furthermore we check the correctness of the sort structure as above.
Since, in this case, $y_1, \ldots, y_n$ are bound by the input prefix, all assumptions on these names are removed from the type assignment, and their monoid elements are added to the rest of the type assignment with $\oplus$.

(T-RESTR) If a name is hidden, we remove the assumption on it, but retain information on its capabilities by adding its monoid element to the type assignment and by keeping the sort.

(T-REP) In this rule we have to make sure that a replicated process has a type assignment which is either idempotent or gets smaller when added to itself. This can be achieved if Γ contains only negative or idempotent monoid elements.
And furthermore, since we know that infinitely many copies of the input prefix with subject x are available, we add $\bot$, $\top$ or 0, according to the value of *in*.

The type system satisfies the following substitution lemma, which is central for proving the subject reduction property:

Lemma 1. (Substitution) *Let $x \neq y$ be two names.*
If $\Gamma, x : \langle s/a \rangle, y : \langle s/b \rangle \vdash P$, then $\Gamma, x : \langle s/a + b \rangle \vdash P\{x/y\}$.

Remark: the proofs can be found in the extended version of this paper [11].

The types defined in table 3 are not yet invariant under reduction: rather than Γ, a modified type assignment $\overline{\Gamma}$ satisfies the subject reduction property.

Let $\Gamma = ob; m; \tilde{x} : \langle \tilde{s}/\tilde{a} \rangle$ and define $\overline{\Gamma} = (ob; m; \tilde{x} : \langle \tilde{s}/\tilde{0} \rangle)^{\langle \tilde{s}, \tilde{a} \rangle}$. That is we add all monoid elements of the remaining free names to m with $\oplus$. Constructing $\overline{\Gamma}$ directly during the typing process does not seem to be possible, since we first have to sum up monoid elements with $+$ and then add them to the type tree with $\oplus$ the moment they are hidden.

Proposition 1. (Subject Reduction Property) *If $P \equiv Q$ and $\Gamma \vdash P$, then it holds also that $\Gamma \vdash Q$. And if $P \rightarrow P'$ and $\Gamma \vdash P$ then there exists a type assignment Γ' such that $\Gamma' \vdash P'$ and $\overline{\Gamma'} \leq \overline{\Gamma}$.*

4 Using the Type System for Process Analysis

As in other type systems for mobile processes, a type guarantees absence of runtime errors which may appear in the form of arity mismatches in the communication rules (R-COMM) and (R-REP), but it also enables us to perform more detailed process analysis.

4.1 Process Capabilities

The aim of this paper is to construct type systems yielding useful results for the analysis and verification of parallel processes. In our case the generic type system gives information concerning structural properties of a process, especially concerning its input and

output capabilities. We will now formally define the connection between the type of a process and its capabilities.

Let P be a process and let x be a free name occurring in P. We define P's *capability* wrt. x by adding the following monoid elements: for every use of x as an output port we add *out* and for every use of x as an input port we add *in*. Notice that we do not continue summation after prefixes (see table 4).

$$C_x(\mathbf{0}) = 0 \quad C_x(P \mid Q) = C_x(P) + C_x(Q)$$

$$C_x(\bar{z}\langle\tilde{y}\rangle.P) = \begin{cases} \mathit{out} \text{ if } x = z \\ 0 \quad \text{otherwise} \end{cases} \quad C_x(z(\tilde{y}).P) = \begin{cases} \mathit{in} \text{ if } x = z \\ 0 \text{ otherwise} \end{cases}$$

$$C_x(!z(\tilde{z}).P) = \begin{cases} \mathit{sig}(\mathit{in}) \text{ if } x = z \\ 0 \quad \text{otherwise} \end{cases} \quad C_x((\nu y\colon s)P) = \begin{cases} C_x(P) \text{ if } x \neq y \\ 0 \quad \text{otherwise} \end{cases}$$

Table 4. Determining the capabilities of a process

Proposition 2. *If $\Gamma \vdash P$, $P \to P'$ and x is a free name of P it follows that $C_x(P') \leq m_{\overline{\Gamma}}(\mathit{sort}_{\Gamma}(x))$, i.e. we determine the sort of x and look up the corresponding monoid element in $\overline{\Gamma}$. And if P contains a subexpression $(\nu y\colon s')Q$ it follows that the capabilities of y will never exceed $m_{\overline{\Gamma}}(s')$.*

4.2 Type Inference

In order to support our claim that the type system is useful for the automated analysis of processes, we roughly sketch a type inference algorithm, determining the smallest type (in the $\leq$ relation defined in section 3) of a process P, provided P has a type. In order to make sure that a smallest type exists, we impose the following condition on the l-monoid: for every monoid element $a \in I$ there is a smallest element a' such that $a \leq a'$ and $a' + a' \leq a'$ (the same must be true for the operation $\oplus$)[1].

The algorithm proceeds in two steps:

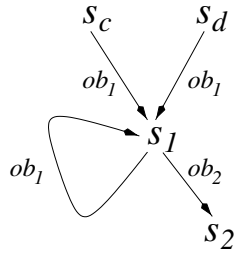
- In the first step we determine the assignment of sorts to names and the mapping ob_{Γ}. This may be done by representing *ob* as a graph and refining *ob* step by step by collapsing graph nodes every time we encounter a constraint of the form $ob(s) = \tilde{s}$. Or we can use the sort inference algorithm by Simon Gay [5].
- In the second step we compute the monoid elements by induction on the structure of P. In this case the typing rules are already very constructive, the main complication arises from typing rule (T-REP). Here we require that the monoid I satisfies the condition stated above. So (because of rule (T-$\leq$)) we may replace every monoid element a with its corresponding a' in the type assignment that we have derived so far.

A straightforward implementation of the algorithm has a runtime complexity quadratic in the size of P. Ameliorations are certainly possible by using efficient algorithms for unification and by finding an intelligent strategy for computing the monoid elements.

[1] Every l-monoid useful for process analysis that we have come across so far satisfies this condition. In the case of $\mathbb{Z}^{\infty}$, a' is ∞ for positive a and a itself for all other elements.

5 Examples

We now get back to the two example processes P and P' introduced in section 2.1 and type them with several instantiations of our type system, and thereby show how to mechanise process analysis in these cases.



We use the algorithm presented in section 4.2 to derive a type assignment Γ for P and P' and in the first step obtain a sort structure ob_Γ as shown in the figure to the left (ob_Γ is the same for P and P'). If there is an arrow labelled ob_i from sort s to sort t, then t is the i-th element of the sequence $ob_\Gamma(s)$. The assignment of names (in brackets we give the bound names) to sorts is:

$$c\colon s_c \quad (d\colon s_d) \quad h, i_1(, r, a, s, h_1)\colon s_1 \quad i_2(, h_2)\colon s_2$$

In the second step the monoid elements $m_{\overline{\Gamma}}(s)$ are computed (see below) in order to give an upper bound for all names having sort s.

5.1 Input/Output Behaviour of Channels

One simple application of our type system is to check whether channels are used for input, output or for both. We use the monoid IO (with elements *none*, *O*–"output only", *I*–"input only" and *both*) introduced in section 2.2. We set $in = I$, $out = O$.

For both processes P and P' we obtain the same type assignments with monoid elements shown in table 5 (row 1), i.e. i_2, h_2 are used neither for input nor output while all other names may be used for both. Note that, because of residuation, typing F alone yields capability *I* for name c, but no output capability. c acquires output capability only if communication with the environment is taking place.

This type system is similar to the one in [15] (apart from the fact that we consider types as a representation of process capabilities, rather than constraints on the environment), our type system however lacks a concept of co- and contravariance and thus our bounds are less tight.

5.2 Upper Bounds on the Number of Active Channels

We attempt to define a type system, similar to the one presented in [8] for our framework, i.e. we want to check how often a channel is used either for input or output.

We use the l-monoid $\mathbb{Z}^\infty \times \mathbb{Z}^\infty$ (cartesian product of the set of integers with ∞ and $-\infty$) introduced in section 2.2. The first component represents the number of active output prefixes (with a fixed subject) and the second component represents the number of active input prefixes.

We set $out = (1, 0)$, $in = (0, 1)$, and typing the processes P and P' yields the results given in table 5 (rows 2 & 3). Since for P the upper bound is always $(1, 1)$ or smaller we can conclude that there is at most one active input port and one active output port for any given subject at a time. For P' we can guarantee that, e.g. $\bar{c}$ always occurs at most once as an output prefix, although it occurs under a replication (see monoid element $m_{\overline{\Gamma}}(s_c)$).

	Property to be checked	$m_{\overline{\Gamma}}(s_d)$	$m_{\overline{\Gamma}}(s_c)$	$m_{\overline{\Gamma}}(s_1)$	$m_{\overline{\Gamma}}(s_2)$
1	Input/Output behaviour of P and P'	*both*	*both*	*both*	*none*
2	Upper bounds on active channels in P	$(1,1)$	$(1,1)$	$(1,1)$	$(0,0)$
3	Upper bounds on active channels in P'	(∞,∞)	$(1,\infty)$	$(1,\infty)$	$(0,0)$
4	Lower bounds on active channels in P	$(-1,0)$	$(-1,0)$	$(-1,-1)$	$(0,0)$
5	Lower bounds on active channels in P'	$(-\infty,\infty)$	$(-\infty,\infty)$	$(-\infty,-1)$	$(0,0)$
6	Avoiding blocked output prefixes in P'	(∞,∞)	$(1,\infty)$	$(1,-1)$	$(0,0)$

Table 5. Resulting monoid elements for different instantiations of the generic type system

5.3 Confluence

As in [8] we can use upper bounds on the number of active channels to guarantee confluence for π-calculus processes (see also [14]). Let Q be a process, and for every name x in Q which is either free or bound by the scope operator ν it holds that its capabilities never exceed $(1,1)$. Then we can guarantee that every channel (also bound channels) occurs at most once at any given time as active input and output prefix, and we have non-overlapping redexes in (R-COMM). Thus we can conclude that if $Q \to^* Q'$, $Q' \to Q_1$ and $Q' \to Q_2$, then either $Q_1 \equiv Q_2$ or there is a process Q_3 such that $Q_1 \to Q_3$ and $Q_2 \to Q_3$.

Row 2 in table 5 provides upper bound $(1,1)$ for all capabilities in P. So we can state that P is confluent. Note that the same process would not be recognised as confluent by the type system in [8].

5.4 Lower Bounds on the Number of Active Channels

The type system is not limited to statements of the form: "there *at most* n active channels", we can also guarantee that there are *at least* m active channels. In order to achieve this, we use the type system above and just invert the partial order, i.e. we take $\geq$ instead of $\leq$, *out* and *in* remain unchanged. This means also that the join $\vee$ in the new partial order is now the meet $\wedge$ of the original partial order. Typing P does not give us much information, since we cannot guarantee that there are at least $m > 0$ prefixes active at any given time (see table 5, row 4) for any channel. In fact, some lower bounds are even (-1) stating that the respective channel removes input (or output) prefixes instead of making them available. In this case $P \to^* \mathbf{0}$ which means that no lower bounds can be guaranteed.

Typing P' yields the monoid elements given in table 5 (row 5) which states that input prefixes with subjects c, d are available infinitely often.

5.5 Avoiding Blocked Prefixes

Another interesting feature is to avoid blocked prefixes, i.e. prefixes which are waiting for a non-existing communication partner. We will first define—with the help of a lattice-ordered monoid—what it means for an output prefix to be blocked.

We take $\mathbb{Z}^\infty \times \mathbb{Z}^\infty$ as an l-monoid and define a new partial order: $(i,j) \sqsubseteq (i',j')$ iff $i \leq i'$ and $j \geq j'$. The first component represents the number of output prefixes and the

second the number of input prefixes of the same subject. *out* $= (1, 0)$ and *in* $= (0, 1)$. We say a name x is *blocking* in P, if $P \rightarrow^* P'$, $C_x(P') \sqsupseteq (1, 0)$ (i.e. there is at least one output prefix with subject x and no corresponding input prefixes) and for all P'' with $P' \rightarrow^* P''$ it follows that $C_x(P'') \sqsupseteq (1, 0)$ (no communication with x will ever take place).

We can, e.g., avoid this situation, by demanding that it is always the case that $C_x(P') = (a, b)$ and either $a \leq 0$ or $b \geq 1$ (i.e. $(a, b) \not\sqsupseteq (1, 0)$). We take the l-monoid and *out*, *in* introduced above. This type system can be obtained by composing a type system establishing upper bounds for input prefixes and one establishing lower bounds for output prefixes. In this way we find out that all output prefixes with subjects c and d are non-blocking in P'.

6 Conclusion and Future Work

This work has a similar aim as that of Honda [6], in that it attempts to describe a general framework for process analysis using type systems. We concentrate on a more specialised but still generic type system, which enables us to prove the subject reduction property for the general case. We have shown that, despite its generality, the type system can be instantiated in order to yield type systems related to existing ones. We have also shown how to parameterise type systems and what kind of parameters are feasible (in our case an l-monoid).

Another type system that has close connections to ours is the linear type system by Kobayashi, Pierce and Turner [8], since it also involves the typing of input/output capabilities of processes. Apart from the more general approach, one new feature of our type system is the introduction of residuation which allows us to recognise the process P in section 5 as confluent, in contrast to the type system in [8]. In some other cases however, our bounds are less tight. The central aim of [8] is to introduce a new notion of barbed congruence by reducing the possible contexts of a process. This question has not been addressed in this paper, it is an interesting direction for future work. For a more detailed discussion of the relation between the two type systems see the full version of this paper [11].

Our type system was derived from a type system for a graph-based process calculus with graphs as types, which make it easier to add additional behaviour information and which have a clear correspondence to associated monoid elements (via morphisms and categorical functors) [9]. A graph-based type system with lattices instead of monoids was presented in [10]. For lattices or positive cones of l-monoids, generic type systems are much easier to present. The main complication arises from non-positive elements and residuation.

Inspiration for this work came from papers deriving information on the behaviour of a process by inspecting its input/output capabilities, such as [15, 14, 8]. In order to conduct process analysis concerning more complex properties (as was done e.g. in [7, 4]) it is necessary to use type systems assigning behaviour information (i.e. monoid elements in our case) not only to single channels, but rather to tuples of channels or other more complex structures. This normally results in a semi-additive type system, in the terminology of Honda [6], while our present type system is strictly additive. In order

to extend this type system, a first solution would be to allow monoid labels for n-ary tuples of names. Another idea is to integrate it into the categorical framework presented in [10], which would allow us to specify very general behaviour descriptions.

We believe that generic type systems can be developed into tools suitable for fast debugging and the analysis of concurrent programs. The next step is to apply the type system presented here to "real-life examples" and to more realistic programming languages.

Acknowledgements: I would like to thank the anonymous referees for their helpful comments, especially for the suggestion to use a sort system instead of type trees.

References

1. Martín Abadi. Secrecy by typing in security protocols. In *Theoretical Aspects of Computer Software*, pages 611–638. Springer-Verlag, 1997.
2. Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.
3. G. Birkhoff. *Lattice Theory*. American Mathematical Society, third edition, 1967.
4. Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Control flow analysis for the pi-calculus. In *Proc. of CONCUR '98*, pages 84–98. Springer-Verlag, 1998. LNCS 1466.
5. Simon J. Gay. A sort inference algorithm for the polyadic π-calculus. In *Proc. of POPL '93*. ACM, 1993.
6. Kohei Honda. Composing processes. In *Proc. of POPL'96*, pages 344–357. ACM, 1996.
7. Naoki Kobayashi. A partially deadlock-free typed process calculus. In *Proc. of LICS '97*, pages 128–139. IEEE, Computer Society Press, 1997.
8. Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. In *Proc. of POPL'96*, pages 358–371. ACM, 1996.
9. Barbara König. *Description and Verification of Mobile Processes with Graph Rewriting Techniques*. PhD thesis, Technische Universität München, 1999.
10. Barbara König. Generating type systems for process graphs. In *Proc. of CONCUR '99*, pages 352–367. Springer-Verlag, 1999. LNCS 1664.
11. Barbara König. Analysing input/output-capabilities of mobile processes with a generic type system (extended version). Technical Report TUM-I0009, Technische Universität München, 2000.
12. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes. *Information and Computation*, 100(1):1–77, 1992.
13. Robin Milner. The polyadic π-calculus: a tutorial. In F. L. Hamer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer-Verlag, Heidelberg, 1993.
14. Uwe Nestmann and Martin Steffen. Typing confluence. In *Second International ERCIM Workshop on Formal Methods in Industrial Critical Systems (Cesena, Italy, July 4–5, 1997)*, pages 77–101, 1997.
15. Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Proc. of LICS '93*, pages 376–385, 1993.
16. James Riely and Matthew Hennessy. Distributed processes and location failures. In *Proc. of ICALP'97*, pages 471–481. Springer-Verlag, 1997. LNCS 1256.

Information Flow vs. Resource Access in the Asynchronous Pi-Calculus (Extended Abstract)

Matthew Hennessy[1] and James Riely[2]

[1] COGS, University of Sussex, UK
email:matthewh@cogs.sussex.ac.uk,
home page: http://www.cogs.susx.ac.uk/users/matthewh/
[2] DePaul University, US
email:jriley@cs.depaul.edu
home page: http://www.depaul.edu/~jriely/

Abstract. We propose an extension of the asynchronous π-calculus in which a variety of security properties may be captured using types. These are an extension of the Input/Output types for the π-calculus in which I/O capabilities are assigned specific security levels.

We define a typing system which ensures that processes running at security level σ cannot access resources with a security level higher than σ. The notion of access control guaranteed by this system is formalized in terms of a Type Safety theorem.

We then show that, for a certain class of processes, our system prohibits implicit information flow from high-level to low-level processes. We prove that low-level behaviour can not be influenced by changes to high-level behaviour. This is formalized as a Non-Interference Theorem with respect to may testing.

1 Introduction

The problem of protecting information and resources in systems with multiple sensitivity or security levels, [2], has been studied extensively. Flow analysis techniques have been used in [3, 4], axiomatic logic in [13] while in [27, 15] type systems have been developed for a number of prototypical programming languages. In this paper, we explore the extent to which type systems for ensuring various forms of security can also be developed for the asynchronous π-calculus [5, 17]. We discuss two security issues: resource access control and information control. The former is described in terms of runtime errors, the latter in terms of non-interference [27, 11].

The (asynchronous) π-calculus is a very expressive language for describing distributed systems, [5, 23, 12], in which processes intercommunicate using channels. Thus $n?(x)\,P$ is a process which receives some value on the channel named n, binds it to the variable x and executes the code P. Corresponding to this input command is the asynchronous output command $n!\langle v\rangle$ which outputs the value v on n. The set of values which may be transmitted on channels includes channel names themselves; this, together with the ability to dynamically create new channel names, gives the language its descriptive power.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 415-427, 2000
© Springer-Verlag Berlin Heidelberg 2000

Within the setting of the π-calculus we wish to investigate the use of types to enforce security policies. To facilitate the discussion we extend the syntax with a new construct to represent a process running at a given security clearance, $\sigma[\![P]\!]$. Here σ is some security level taken from a complete lattice of security levels SL and P is the code of the process. Further, we associate with each channel, the *resources* in our language, a set of input/output capabilities [22, 24], each decorated with a specific security level. Intuitively, if channel n has a read capability at level σ, then only processes running at security level σ or higher may be read from n. This leads to the notion of a *security policy* Σ, which associates a set of capabilities with each channel in the system. The question then is to design a typing system which ensures that processes do not violate the given security policy.

Of course this depends on when we consider such a violation to take place. For example if Σ assigns the channel or resource n the highest security level top then it is reasonable to say that a violation will eventually occur in

$$c!\langle n\rangle \quad | \quad \mathsf{bot}[\![c?(x)\ x?(y)\,P]\!]$$

as after the communication on c, a low level process, $\mathsf{bot}[\![n?(y)\,P]\!]$ has gained access to the high level resource n. Underlying this example is the principle that processes at a given security level σ should have access to resources at security level *at most* σ. We formalize this principle in terms of a relation $P \stackrel{\Sigma}{\mapsto} err$, indicating that P violates the security policy Σ.

To prevent such errors, we restrict attention to security policies that are somehow consistent. Let Γ be such a consistent policy; consistency is defined by restricting types so that they respect a subtyping relation. We then introduce a typing system, $\Gamma \vdash P$, which ensures that P can never violate Γ:

> If $\Gamma \vdash P$ then for every context $C[\ ]$ such that $\Gamma \vdash C[P]$ and every Q which occurs during the execution of $C[P]$, that is $C[P] \mapsto^* Q$, we have $Q \not\stackrel{\Gamma}{\mapsto} err$.

Thus our typing system ensures that low level processes will never gain access to high level resources. The typing system implements a particular view of security, which we refer to as the *R-security policy*, as it offers protection to *resources*. Here communication is allowed between high level and low level principals, provided of course that the values involved are appropriate.

This policy does not rule out the possibility of information leaking indirectly from high security to low security principals. Suppose h is a high channel and hl is a channel with high-level write access and low-level read access in:

$$\mathsf{top}[\![h?(x)\,\mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{hl}!\langle 0\rangle\ \mathsf{else}\ \mathsf{hl}!\langle 1\rangle]\!] \mid \mathsf{bot}[\![\mathsf{hl}?(z)\,Q]\!] \qquad (\star)$$

This system can be well-typed although there is some implicit information flow from the high security agent to the low security one; the value received on the high level channel h can be determined by the low level process Q.

It is difficult to formalize exactly what is meant by *implicit information flow* and in the literature various authors have instead relied on *non-interference*, [14,

25, 11, 26], a concept more amenable to formalization, which ensures, at least informally, the absence of implicit information flow.

To obtain such results for the π-calculus we need, as the above example shows, a stricter security policy, which we refer to as the *I-security policy*. This allows a high level principal to read from low level resources but not to write to them. Using the terminology of [2, 7]:

- *write up*: a process at level σ may only write to channels at level σ or above
- *read down*: a process at level σ may only read from channels at level σ or below.

In fact the type inference system remains the same and we only need constrain the notion of type. In this restricted type system well-typing, $\Gamma \Vdash P$, ensures a form of *non-interference*.

To formalize this non-interference result we need to develop a notion of process behaviour, relative to a given security level. Since the behaviour of processes also depends on the type environment in which they operate we need to define a relation

$$P \approx^{\sigma}_{\Gamma} Q$$

which intuitively states that, relative to Γ, there is no observable distinction between the behaviour of P and Q at security level σ; processes running at security level σ can observe no difference in the behaviour of P and Q. Lack of information flow from high to low security levels now means that this relation is invariant under changes in high-level values; or indeed under changes in high-level behaviour.

It turns out that the extent to which this is true depends on the exact formulation of the behavioural equivalence $\approx^{\sigma}_{\Gamma}$. We show that it is *not* true if $\approx^{\sigma}_{\Gamma}$ is based on *observational* equivalence [19] or *must testing* equivalence [21]. But a result can be established if we restrict our attention to *may testing* equivalence (here written $\simeq^{\sigma}_{\Gamma}$). Specifically we will show that, for certain H, K:

$$\text{If } \Gamma \Vdash^{\mathsf{g}} P,\ Q \text{ and } \Gamma \Vdash^{\mathsf{top}} H,\ K \text{ then } P \simeq^{\sigma}_{\Gamma} Q \text{ implies } P \,|\, H \simeq^{\sigma}_{\Gamma} Q \,|\, K \quad (\star\star)$$

The remainder of the paper is organized as follows. In the next section we define the *security π-calculus*, giving a labelled transition semantics and a formal definition of runtime errors. In Section 3 we design a set of types and a typing system which implements the resource control policy. This section also contains Subject Reduction and Type Safety theorems. In Section 4 we motivate the restrictions required on types and terms in order to implement the information control policy. We also give a precise statement of our non-interference result, and give counter-examples to related conjectures based on equivalences other than *may testing*.

The proof of our main theorem depends on an analysis of *may testing* in terms of *asynchronous* sequences of actions [6] which in turn depends on a more explicit operational semantics for our language, where actions are paramterised relative to a typing environment. The details may be found in the full version of the paper, [16].

Fig. 1 Syntax

P, Q ::=	*Terms*	X, Y ::=	*Patterns*
$u!\langle v\rangle$	Output	x	Variable
$u?(X:\mathrm{A})\,P$	Input	$(X_1, \ldots, X_k)$	Tuple
$\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Matching		
$\sigma[\![P]\!]$	Security level	u, v, w ::=	*Values*
$(\mathsf{new}\, a:\mathrm{A})\ P$	Name creation	bv	Base Value
$P \mid Q$	Composition	a	Name
$*P$	Replication	x	Variable
$\mathbf{0}$	Termination	$(u_1, \ldots, u_k)$	Tuple

2 The Language

The syntax of the *security π-calculus*, given in Figure 1, uses a predefined set of *names*, ranged over by $a, b, \ldots, n$ and a set of *variables*, ranged over by x, y, z. *Identifiers* are either variables or names. *Security annotations*, ranged over by small Greek letters $\sigma, \rho, \ldots$, are taken from a complete lattice $\langle SL, \preceq, \sqcap, \sqcup, \mathsf{top}, \mathsf{bot}\rangle$ of security levels. We also assume for each σ a set of *basic values* BV_σ; we use bv to range over base values. We require that all syntactic sets be disjoint.

The binding constructs $u?(X:\mathrm{A})\,Q$ and $(\mathsf{new}\, a:\mathrm{A})\ Q$ introduce the usual notions of free names and variables, $\mathsf{fn}(P)$ and $\mathsf{fv}(P)$, respectively, and associated notions of substitution; details may be found in the full version. Moreover the typing annotations on the binding constructs, which will be explained in Section 3, are omitted whenever they do not play a role.

The behaviour of a process is determined by the interactions in which it can engage. To define these, we give a labelled transition semantics (LTS) for the language. The set *Act* of *labels*, or *actions*, is defined as follows:

μ ::=	*Actions*
τ	Internal action
$(\tilde{c}:\tilde{\mathrm{C}})a?v$	Input of v on a learning private names $\tilde{c}$
$(\tilde{c}:\tilde{\mathrm{C}})a!v$	Output of v on a revealing private names $\tilde{c}$

Visible actions (all except τ) are ranged over by α, β and we use $\mathcal{E}(\alpha)$ to denote the bound names in α, together with their types. $\mathcal{E}((\tilde{c}:\tilde{\mathrm{C}})a!v) = \mathcal{E}((\tilde{c}:\tilde{\mathrm{C}})a?v) = (\tilde{c}:\tilde{\mathrm{C}})$. Further, let $\mathsf{n}(\mu)$ be the set of *names* occurring in μ, whether free or bound. We say that the actions '$(\tilde{c}:\tilde{\mathrm{C}})a?v$' and '$(\tilde{c}:\tilde{\mathrm{C}})a!v$' are *complementary*, with $\overline{\alpha}$ denoting the complement of α.

The LTS is defined in Figure 2 and for the most part the rules are straightforward; it is based on the standard operational semantics from [20], to which the reader is referred for more motivation.

Informally a security policy associates with each input/output capability on a channel a security level. To this end, *Pre-capabilities* and *pre-types* are defined

Fig. 2 Labelled Transition Semantics

(L-OUT)

$$\frac{}{a!\langle v\rangle \xrightarrow{a!v} \mathbf{0}}$$

(L-IN)

$$\frac{}{a?(X)\,P \xrightarrow{(\tilde{c}:\mathrm{C})a?v} P\{\!\{^{v}/_{X}\}\!\}} \quad \tilde{c} \notin \mathsf{fn}(P), \tilde{c} \in \mathsf{fn}(v)$$

(L-OPEN)

$$\frac{P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} P'}{(\mathsf{new}\, b:\mathrm{B})\ P \xrightarrow{(b:\mathrm{B})(\tilde{c}:\mathrm{C})a!v} P'} \quad \begin{array}{l} b \neq a \\ b \in \mathsf{fn}(v) \end{array}$$

(L-COM)

$$\frac{P \xrightarrow{\alpha} P',\ \ Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} (\mathsf{new}\, \mathcal{E}(\alpha))\ (P' \mid Q')}$$

(L-EQ)

$$\frac{}{\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P} \qquad \frac{}{\mathsf{if}\ u = w\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q} \quad u \neq w$$

(L-CTXT)

$$\frac{P \xrightarrow{\mu} P'}{\begin{array}{l} *P \xrightarrow{\mu} *P \mid P' \\ \sigma[\![P]\!] \xrightarrow{\mu} \sigma[\![P']\!] \end{array}} \qquad \frac{P \xrightarrow{\mu} P'}{\begin{array}{l} P \mid Q \xrightarrow{\mu} P' \mid Q \\ Q \mid P \xrightarrow{\mu} Q \mid P' \end{array}} \quad \mathsf{bn}(\mu) \notin \mathsf{fn}(Q)$$

$$\frac{P \xrightarrow{\mu} P'}{(\mathsf{new}\, a:\mathrm{A})\ P \xrightarrow{\mu} (\mathsf{new}\, a:\mathrm{A})\ P'} \quad a \notin \mathsf{n}(\mu)$$

as follows:

cap ::=	*Pre-Capability*
$\mathsf{w}_\sigma\langle \mathrm{A}\rangle$	σ-level process can write values with type A
$\mathsf{r}_\sigma\langle \mathrm{A}\rangle$	σ-level process can read values with type A
A ::=	*Pre-Type*
$\mathbf{B}_\sigma$	Base type
$\{cap_1, \ldots, cap_k\}$	Resource type ($k \geq 0$)
$(\mathrm{A}_1, \ldots, \mathrm{A}_k)$	Tuple type ($k \geq 0$)

We will tend to abbreviate a singleton set of capabilities, $\{cap\}$, to *cap*.

A *security policy*, Σ, is a finite mapping from names to pre-types. Thus, for example, if Σ maps the channel lh to the pre-type $\{\mathsf{w}_{\mathsf{bot}}\langle \mathrm{B}\rangle,\ \mathsf{r}_{\mathsf{top}}\langle \mathrm{A}\rangle\}$, for some appropriate A, B, then low level processes may write to lh but only high level ones may read from it; this is an approximation of the security associated with a mailbox. On the other hand if Σ maps hl to $\{\mathsf{r}_{\mathsf{bot}}\langle \mathrm{A}\rangle,\ \mathsf{w}_{\mathsf{top}}\langle \mathrm{B}\rangle\}$ then hl acts more like an information channel; anybody can read from it but only high level processes may place information there.

The import of a security policy may be underlined by defining what it means to violate it. Our definition is given in Figure 3, in terms of a relation $P \overset{\Sigma}{\longmapsto} err$. For example, relative to the policy Σ defined above, after one reduction step of the process $\mathsf{top}[\![c!\langle \mathsf{hl}\rangle]\!] \mid \mathsf{bot}[\![c?(x)\, x!\langle v\rangle]\!]$, there is a security error because

Fig. 3 Runtime Errors

$$
\begin{array}{lll}
(\text{E-RD}) & \rho[\![a?(X)\,P]\!] \xmapsto{\Sigma} \mathit{err} & \text{if } \sigma \preceq \rho \text{ implies for all A, } \mathsf{r}_\sigma\langle \mathrm{A}\rangle \notin \Sigma(a) \\
(\text{E-WR}_1) & \rho[\![a!\langle v\rangle]\!] \xmapsto{\Sigma} \mathit{err} & \text{if } \sigma \preceq \rho \text{ implies for all A, } \mathsf{w}_\sigma\langle \mathrm{A}\rangle \notin \Sigma(a) \\
(\text{E-WR}_2) & \rho[\![a!\langle v\rangle]\!] \xmapsto{\Sigma} \mathit{err} & \text{if } bv \in v,\ bv \in \mathbf{B}_\sigma \text{ and } \sigma \not\preceq \rho
\end{array}
$$

$$
(\text{E-STR})\quad \frac{P \xmapsto{\Sigma} \mathit{err}}{P \mid Q \xmapsto{\Sigma} \mathit{err}} \qquad \frac{P \xmapsto{\Sigma} \mathit{err}}{\rho[\![P]\!] \xmapsto{\Sigma} \mathit{err}} \qquad \frac{P \equiv Q,\ P \xmapsto{\Sigma} \mathit{err}}{Q \xmapsto{\Sigma} \mathit{err}}
$$

$$
\frac{P \xmapsto{\Sigma, a:\mathrm{A}} \mathit{err}}{(\mathsf{new}\ n : \mathrm{A})\ P \xmapsto{\Sigma} \mathit{err}}
$$

$\mathsf{bot}[\![\mathsf{hl}!\langle v\rangle]\!] \xmapsto{\Sigma} \mathit{err}$. A low security process has read access to security channel hl on which write access is reserved for high-security processes. Assuming an appropriate typing for c and v the same security error does not occur in $\mathsf{top}[\![c!\langle \mathsf{lh}\rangle]\!] \mid \mathsf{bot}[\![c?(x)\,x!\langle v\rangle]\!]$. The low security process $\mathsf{bot}[\![\mathsf{lh}!\langle v\rangle\, Q]\!]$ has the right to write on the channel lh.

3 Resource Control

Our typing system will apply only to certain security policies, those in which the pre-types are in some sense *consistent*. Consistency is imposed using a system of kinds: the kind RType_σ comprises the value types accessible to processes at security level σ. These kinds are in turn defined using a subtyping relation on pre-capabilities and pre-types.

Definition 1. *Let* $<:$ *be the least preorder on pre-capabilities and pre-types such that:*

$$
\begin{array}{lrll}
(\text{U-WR}) & \mathsf{w}_\sigma\langle \mathrm{A}\rangle <: \mathsf{w}_\sigma\langle \mathrm{B}\rangle & \textit{if } \mathrm{B} <: \mathrm{A} \\
(\text{U-RD}) & \mathsf{r}_\sigma\langle \mathrm{A}\rangle <: \mathsf{r}_\rho\langle \mathrm{B}\rangle & \textit{if } \mathrm{A} <: \mathrm{B} \textit{ and } \sigma \preceq \rho \\
(\text{U-BASE}) & \mathbf{B}_\sigma <: \mathbf{B}_\rho & \textit{if } \sigma \preceq \rho \\
(\text{U-RES}) & \{\mathrm{cap}_i\}_{i\in I} <: \{\mathrm{cap}'_j\}_{j\in J} & \textit{if } (\forall j)(\exists i)\ \mathrm{cap}_i <: \mathrm{cap}'_j \\
(\text{U-TUP}) & (\mathrm{A}_1, \ldots, \mathrm{A}_k) <: (\mathrm{B}_1, \ldots, \mathrm{B}_k) & \textit{if } (\forall i)\ \mathrm{A}_i <: \mathrm{B}_i
\end{array}
$$

For each ρ*, let* RType_ρ *be the least set that satisfies:*

$$
(\text{RT-WR})\quad \frac{\mathrm{A} \in \mathrm{RType}_\sigma}{\{\mathsf{w}_\sigma\langle \mathrm{A}\rangle\} \in \mathrm{RType}_\rho}\ \sigma \preceq \rho
$$

$$
(\text{RT-RD})\quad \frac{\mathrm{A} \in \mathrm{RType}_\sigma}{\{\mathsf{r}_\sigma\langle \mathrm{A}\rangle\} \in \mathrm{RType}_\rho}\ \sigma \preceq \rho
$$

$$
(\text{RT-BASE})\quad \frac{}{\mathbf{B}_\sigma \in \mathrm{RType}_\rho}\ \sigma \preceq \rho
$$

$$
(\text{RT-WRRD})\quad \frac{\begin{array}{l}\mathrm{A} \in \mathrm{RType}_\sigma \\ \mathrm{A}' \in \mathrm{RType}_{\sigma'}\end{array}}{\{\mathsf{w}_\sigma\langle \mathrm{A}\rangle, \mathsf{r}_{\sigma'}\langle \mathrm{A}'\rangle\} \in \mathrm{RType}_\rho}\ \begin{array}{l}\sigma \preceq \rho \\ \sigma' \preceq \rho \\ \mathrm{A} <: \mathrm{A}'\end{array}
$$

$$
(\text{RT-TUP})\quad \frac{\mathrm{A}_i \in \mathrm{RType}_\rho \quad (\forall i)}{(\mathrm{A}_1, \ldots, \mathrm{A}_k) \in \mathrm{RType}_\rho}
$$

Fig. 4 Typing Rules

(T-ID)
$$\frac{\Gamma(u) <: \mathrm{A}}{\Gamma \vdash u : \mathrm{A}}$$

(T-BASE)
$$\frac{bv \in \mathbf{B}_\sigma}{\Gamma \vdash bv : \mathbf{B}_\sigma}$$

(T-TUP)
$$\frac{\Gamma \vdash v_i : A_i \quad (\forall i)}{\Gamma \vdash (v_1, \ldots, v_k) : (\mathrm{A}_1, \ldots, \mathrm{A}_k)}$$

(T-IN)
$$\frac{\Gamma, X : \mathrm{A} \Vdash^{\sigma} P \qquad \Gamma \vdash u : \mathsf{r}_\sigma\langle \mathrm{A} \rangle}{\Gamma \Vdash^{\sigma} u?(X : \mathrm{A})\, P}$$

(T-OUT)
$$\frac{\Gamma \vdash u : \mathsf{w}_\sigma\langle \mathrm{A} \rangle \qquad \Gamma \vdash v : \mathrm{A}}{\Gamma \Vdash^{\sigma} u!\langle v \rangle}$$

(T-EQ)
$$\frac{\Gamma \vdash u : \mathrm{A}, v : \mathrm{B} \qquad \Gamma \Vdash^{\sigma} Q \qquad \Gamma \sqcap \{u : \mathrm{B}, v : \mathrm{A}\} \Vdash^{\sigma} P}{\Gamma \Vdash^{\sigma} \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q}$$

(T-SR)
$$\frac{\Gamma \Vdash^{\sigma \sqcap \rho} P}{\Gamma \Vdash^{\sigma} \rho[\![P]\!]}$$

(T-NEW)
$$\frac{\Gamma, a : \mathrm{A} \Vdash^{\sigma} P}{\Gamma \Vdash^{\sigma} (\mathsf{new}\ a : \mathrm{A})\ P}$$

(T-STR)
$$\frac{\Gamma \Vdash^{\sigma} P, Q}{\Gamma \Vdash^{\sigma} P \mid Q, *P, \mathbf{0}}$$

Let RType *be the union of the kinds* RType_ρ *over all* ρ. □

Note that if $\sigma \preceq \rho$ then $RType_\sigma \subseteq RType_\rho$. Intuitively, low level values are accessible to high level processes. However the converse is not true. For example $\mathsf{w}_{\mathsf{top}}\langle\rangle \in RType_{\mathsf{top}}$ but $\mathsf{w}_{\mathsf{top}}\langle\rangle$ is not in $RType_{\mathsf{bot}}$. The compatibility requirement between read and write capabilities in a type (RT-WRRD), in addition to the typing implications discussed in [24], also has security implications. For example suppose $\mathsf{r}_{\mathsf{bot}}\langle \mathbf{B}_\sigma \rangle$ and $\mathsf{w}_{\mathsf{top}}\langle \mathrm{B} \rangle$ are capabilities in a valid channel type. Then apriori a high level process can write to the channel while a low level process may read from it. However the only possibility for σ is bot, that is only low level values may be read. Moreover the requirement $\mathrm{B} <: \mathbf{B}_\sigma$ implies that B must also be $\mathbf{B}_{\mathsf{bot}}$. So although high level processes may write to the channel they may only write low level values.

Proposition 1. *For every* ρ, RType_ρ *is a preorder with respect to* $<:$, *with both a partial meet operation* $\sqcap$ *and a partial join* $\sqcup$. □

A *type environment* is a finite mapping from identifiers (names and variables) to types. We adopt some standard notation. For example, let '$\Gamma, u : \mathrm{A}$' denote the obvious extension of Γ; '$\Gamma, u : \mathrm{A}$' is only defined if u is not in the domain of Γ. The subtyping relation $<:$ together with the partial operators $\sqcap$ and $\sqcup$ may also be extended to environments. We will normally abbreviate the simple environment $\{u : \mathrm{A}\}$ to $u : \mathrm{A}$ and moreover use $v : \mathrm{A}$ to denote its obvious generalisation to values.

The typing system is given in Figure 4 where the judgements are of the form '$\Gamma \Vdash^{\sigma} P$'. If $\Gamma \Vdash^{\sigma} P$ we say that P is a *σ-level process*. Also, let '$\Gamma \vdash P$' abbreviate '$\Gamma \Vdash^{\mathsf{top}} P$'.

Intuitively '$\Gamma \Vdash^{\sigma} P$' indicates that the process P will not cause any security errors if executed with security clearance σ. The rules are very similar to those used in papers such as [24, 22] for the standard IO typing of the π-calculus. Indeed the only significant use of the security levels is in the (T-IN) and (T-OUT)

rules, where the channels are required to have a specific security level. This is inferred using auxiliary value judgements, of the form $\Gamma \vdash v : \mathrm{A}$. It is interesting to note that security levels play no direct role in their derivation.

Theorem 1 (Subject Reduction). *Suppose $\Gamma \models^{\sigma} P$. Then*

- $P \xrightarrow{\tau} Q$ *implies* $\Gamma \models^{\sigma} Q$
- $P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a?v} Q$ *implies there exists a type* A *such that* $\Gamma \vdash a : \mathsf{r}_{\delta}\langle \mathrm{A} \rangle$ *for some* $\delta \preceq \sigma$, *and if* $\Gamma \sqcap v : \mathrm{A}$ *is well-defined then* $\Gamma \sqcap v : \mathrm{A} \models^{\sigma} Q$.
- $P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} Q$ *implies there exists a type* A *such that* $\Gamma \vdash a : \mathsf{w}_{\delta}\langle \mathrm{A} \rangle$ *for some* $\delta \preceq \sigma$, $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \vdash v : \mathrm{A}$ *and* $\Gamma, \tilde{c} : \tilde{\mathrm{C}} \models^{\sigma} Q$.

□

We can now state the first main result:

Theorem 2 (Type Safety). *If $\Gamma \vdash P$ then for every closed context $C[\,]$ such that $\Gamma \vdash C[P]$ and every Q such that $C[P] \xrightarrow{\tau}{}^{*} Q$ we have $Q \overset{\Gamma}{\not\mapsto} \mathrm{err}$*

□

Having defined our typing system we may now view $\sigma[\![P]\!]$ simply as notation for the fact that, relative to the current typing environment Γ, the process P is well-typed at level σ, i.e. $\Gamma \models^{\sigma} P$. Technically we can view $\sigma[\![P]\!]$ to be *structurally equivalent* to P, assuming we are working in an environment Γ such that $\Gamma \models^{\sigma} P$.

4 Information Flow

We have shown in the previous sections that, in well-typed systems, processes running at a given security level can only access resources appropriate to that level. However, as pointed out in the Introduction this does not rule out (implicit) information flow between levels. One way of formalizing this notion of flow of information is to consider the behaviour of processes and how it can be influenced. If the behaviour of low-level processes is independent of any high-level values in its environment then we can say that there can be no implicit flow of information from high-level to low-level. This is *not* the case in the example considered in the Introduction, $(\star)$. Suppose, for example, that Q is the code fragment '$\mathsf{if}\ z = 0\ \mathsf{then}\ l_1!\langle\rangle\ \mathsf{else}\ l_2!\langle\rangle$'. If $(\star)$ were placed in an environment with '$\mathsf{top}[\![h!\langle 0\rangle]\!]$', then the resource l_1 would be called. If, instead, $(\star)$ were placed in an environment with '$\mathsf{top}[\![h!\langle 42\rangle]\!]$', then l_2 would be called. In other words the behaviour of the low-level process can be influenced by high-level changes; there is a possibility of information flow downwards.

This is not surprising in view of the type associated with the channel hl; in the terminology of [2] it allows a *write down* from a high-level process to a low-level process. Thus if we are to eliminate implicit information flow between levels in well-typed processes we need to restrict further the allowed types; types such as $\{\mathsf{w}_{\mathsf{top}}\langle\rangle, \mathsf{r}_{\mathsf{bot}}\langle\rangle\}$ clearly contradict the spirit of secrecy. Thus, for the rest of the paper we work with the more restrictive set *IType*, the *Information types*. In order for $\{\mathsf{w}_{\sigma}\langle \mathrm{A}\rangle, \mathsf{r}_{\sigma'}\langle \mathrm{A}'\rangle\}$ to be in *IType*, it must be that $\sigma \preceq \sigma'$; this is not necessarily true for types in *RType*.

Definition 2. *For each ρ, let* IType_ρ*, be the least set that satisfies the rules in Definition 1, with* (RT-WRRD) *replaced by:*

$$\frac{\begin{array}{l}\mathrm{A} \in \mathrm{IType}_\sigma \\ \mathrm{A}' \in \mathrm{IType}_{\sigma'}\end{array}}{\{\mathsf{w}_\sigma\langle \mathrm{A}\rangle,\ \mathsf{r}_{\sigma'}\langle \mathrm{A}'\rangle\} \in \mathrm{IType}_\rho}\ (\text{IT-WRRD})\quad \begin{array}{l}\sigma \preceq \sigma' \\ \sigma' \preceq \rho \\ \mathrm{A} <: \mathrm{A}'\end{array}$$

Let IType *be the union of* IType_ρ *over all ρ. We write $\Gamma \Vdash^{\sigma} P$ if $\Gamma \vdash^{\sigma} P$ can be derived from the rules of Figure 4 using these more restrictive types.* □

All of the results of the previous section carry over to the stronger typing system; we leave their elaboration to the reader.

Unfortunately, due to the expressiveness of our language, the use of *I-types* still does not preclude information flow downwards, between levels. Consider the system

$$\mathsf{top}\,[\![h?(x)\ \mathsf{if}\ x = 0\ \mathsf{then}\ \mathsf{bot}[\![l!\langle 0\rangle]\!]\ \mathsf{else}\ \mathsf{bot}[\![l!\langle 1\rangle]\!]]\!] \mid \mathsf{bot}\,[\![l?(z)\,Q]\!]$$

executing in an environment in which h is a top-level read/write channel and l is a bot-level read/write channel. This system can be well-typed using *I-types*, but there still appears to be some some implicit flow of information from top to bot. The problem here is that our syntax allows a high-level process, which can not write to low-level channels, to evolve into a low-level process which does have this capability; we need to place a boundary between low- and high-level processes which ensures a high-level process never gains write access to low-level channels. This is the aim of the following definition:

Definition 3. *Define the security levels of a term below ρ,* $\mathrm{sl}_\rho(P)$*, as follows:*

$$\begin{array}{rrr}
\mathrm{sl}_\rho(*P) = \mathrm{sl}_\rho(P) & \mathrm{sl}_\rho(\mathbf{0}) = \{\rho\} & \mathrm{sl}_\rho(\sigma[\![P]\!]) = \{\sigma \sqcap \rho\} \cup \mathrm{sl}_{\sigma\sqcap\rho}(P) \\
\mathrm{sl}_\rho((\mathsf{new}\ a : \mathrm{A})\ P) = \mathrm{sl}_\rho(P) & \mathrm{sl}_\rho(u!\langle v\rangle) = \emptyset & \mathrm{sl}_\rho(P \mid Q) = \mathrm{sl}_\rho(P) \cup \mathrm{sl}_\rho(Q) \\
\mathrm{sl}_\rho(u?(X : \mathrm{B})\ P) = \mathrm{sl}_\rho(P) & \mathrm{sl}_\rho(\mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ Q) = \mathrm{sl}_\rho(P) \cup \mathrm{sl}_\rho(Q) &
\end{array}$$

*A process P is σ-*free *if for every ρ in* $\mathrm{sl}_{\mathsf{top}}(P)$*, $\rho \not\preceq \sigma$.* □

Non-interference, as discussed in the Introduction, ($\star\star$), depends on a formulation of a behavioural equivalence, as the following example illustrates. Let A denote the type $\{\mathsf{w}_{\mathsf{bot}}\langle\rangle,\ \mathsf{r}_{\mathsf{bot}}\langle\rangle\}$ and B denote $\{\mathsf{r}_{\mathsf{bot}}\langle\rangle\}$. Further, let Γ map a and b to A and B, respectively, and n to the type $\{\mathsf{w}_{\mathsf{bot}}\langle \mathrm{A}\rangle, \mathsf{r}_{\mathsf{bot}}\langle \mathrm{A}\rangle\}$. Now consider the terms P and H defined by

$$P \Leftarrow \mathsf{bot}[\![n!\langle a\rangle \mid n?(x : \mathrm{A})\ x!\langle\rangle]\!] \qquad H \Leftarrow \mathsf{top}[\![n?(x : \mathrm{B})\ b?(y)\,\mathbf{0}]\!]$$

It is very easy to check that $\Gamma \Vdash P, H$ and that H is bot-free. Note that in the term $P \mid H$ there is contention between the low and high-level processes for who will receive a value on the channel n. This means that if we were to base the

semantic relation $\approx$ on any of *strong bisimulation equivalence*, *weak bisimulation equivalence*, [19], or *must testing*, [21], we would have

$$P \mid \mathbf{0} \not\approx^{\sigma} P \mid H$$

The essential reason is that the consumption of writes can be detected; the reduction

$$P \mid H \xrightarrow{\tau} \mathsf{bot}[\![n?(x:\mathrm{A})\ x!\langle\rangle]\!] \mid \mathsf{top}[\![b?(y)\,.\,\mathbf{0}]\!]$$

cannot be matched by $P \mid \mathbf{0}$. Using the terminology of [21], $P \mid \mathbf{0}$ *guarantees* the test $\mathsf{bot}[\![a?(x)\,\omega!\langle\rangle]\!]$ whereas $P \mid H$ does not.

May equivalence is defined in terms of tests. A *test* is a process with an occurrence of a new reserved resource name ω. We use T to range over tests, with the typing rule $\Gamma \Vdash^{\mathsf{g}} \omega!\langle\rangle$ for all Γ. When placed in parallel with a process P, a test may interact with P, producing an output on ω if some desired behaviour of P has been observed. We write $T\Downarrow$ if $T \xrightarrow{\tau}{}^{*} T'$, where T' has the form $(\mathsf{new}\ \tilde{c})\ (\omega!\langle\rangle \mid T'')$ for some T'' and $\tilde{c}$; that is T can eventually report success.

We wish to capture the behaviour of processes at a given level of security. Consequently we only compare their ability to pass tests that are well-typed at that level. The definition must also take into account the environment in which the processes are used, as this determines the security level associated with resources.

Definition 4. *We write $P \simeq^{\sigma}_{\Gamma} Q$ if for every test T such that $\Gamma \Vdash^{\mathsf{g}} T$:*

$$(P \mid T)\Downarrow \text{ if and only if } (Q \mid T)\Downarrow$$

□

We can now state the main result of the paper.

Theorem 3 (Non-Interference). *If $\Gamma \Vdash^{\mathsf{g}} P, Q$ and $\Gamma \Vdash^{\mathsf{top}} H, K$ where H and K are σ-free processes, then $P \simeq^{\sigma}_{\Gamma} Q$ implies $P \mid H \simeq^{\sigma}_{\Gamma} Q \mid K$.* □

The proof of the theorem relies on a constructing sufficient condition to guarantee that two processes are may equivalent. This condition involves the *asynchronous* sequences of actions which processes can perform in the type environment Γ. The details may be found in the full version of the paper, [16], which also contains the subsequent proof of the non-interference result.

Finally let us remark that if we allowed *synchronous* tests then this result would no longer hold. For an appropriate Γ would have:

$$P \mid \mathbf{0} \simeq^{\sigma}_{\Gamma} P \mid H$$

Let T be the test $\mathsf{bot}[\![b!\langle\rangle\,\omega!\langle\rangle]\!]$. Then $P \mid H \mid T$ may eventually produce an output on ω whereas $P \mid \mathbf{0} \mid T$ cannot. However, since our language is asynchronous, such tests are not allowed.

5 Conclusions and Related Work

Methods for controling information flow are a central research issue in computer security [7, 14, 27] and in the Introduction we have indicated a number of different approaches to its formalisation. Non-interference has emerged as a useful concept and is widely used to infer (indirectly) the absence of information flow. In publications such as [25, 9] it has been pointed out that process algebras may be fruitfully used to formalise and investigate this concept; for example in [8] process algebra based methods are suggested for investigating security protocols, essentially using a formalisation of non-interference for CCS.

However in these publications the *non-interference* is always defined behaviourally, as a condition on the possible traces of CCS or CSP processes; useful surveys of trace based non-interference may be found in [9, 26]. Here, we work with the more expressive π-calculus, which allows dynamic process creation and network reconfiguration. Our approach to *non-interference* is also more extensional in that it is expressed in terms of how processes effect their environments, relative to a particular behavioural equivalence. However the proof of our main result, Theorem 3, describes may equivalence in terms of (typed) traces; presumably a trace based definition of *non-interference*, similar in style to those in [9, 26] could be extracted from this proof.

More importantly our approach differs from much of the recent process calculus based security research in that we develop purely *static* methods for ensuring security. Processes are shown to be secure not by demonstrating some property of trace sets, using a tool as such as that in [10], but by type-checking. Types have also been used in this manner in [1] for an extension of the π-calculus called the *spi-calculus*. But there the structure of the types are very straightforward; the type *Secret* representing a secret channel, the type *Public* representing a public one, and *Any* which could be either. However the main interest is in the type rules for the encryption/decryption primitives of the *spi-calculus*. The non-interference result also has a different formulation to ours; it states that the behaviour of well-typed processes is invariant, relative to *may* testing, under certain value-substitutions. Intuitively, it means that the encryption/decryption primitives preserve values of type *Secret* from certain kinds of attackers. It would be interesting to add these primitives to the our *security π-calculus* and to try to adapt the associated type rules to the set of *I-Types*.

An extension of the π-calculus is also considered in [18], where a sophisticated type system is used to control information flow. The judgements in their system take the form

$$\Gamma \vdash_s P \triangleright A$$

where s is a security level, P is a process term, A is a poset of so-called *action nodes* and Γ is a type environment. Their environments are quite similar to ours, essentially associating with channels a version of input/output types annotated with, among other things, security levels. However their intuition, and much of the technical development, is quite different from ours. In summary it appears that our type system addresses information flow within the core π-calculus while the more sophisticated one of [18] controls the flow allowed via the extra syntactic

constructs of their language. However a more thorough comparison between the two systems deserves to be made.

Acknowledgements: The research was partially funded by EPSRC grant GR/L93056, and ESPRT Working Group Confer2. The authors would like to thank I. Castellani for a careful reading of a draft version of the paper.

References

1. Martín Abadi. Secrecy by typing in security protocols. In *Proceedings of TACS'97*, volume 1281 of *Lecture Notes in Computer Science*, pages 611–637. Springer Verlag, 1997.
2. D. E. Bell and L. J. LaPadula. Secure computer system: Unified exposition and multics interpretation. Technical report MTR-2997, MITRE Corporation, 1975.
3. C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Control flow analysis for the π-calculus. In *Proc. CONCUR'98*, number 1466 in Lecture Notes in Computer Science, pages 84–98. Springer-Verlag, 1998.
4. C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Static analysis of processes for no read-up and no write-down. In *Proc. FOSSACS'99*, number 1578 in Lecture Notes in Computer Science, pages 120–134. Springer-Verlag, 1999.
5. G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA-Sophia Antipolis, 1992.
6. Ilaria Castellani and Matthew Hennessy. Testing theories for asynchronous languages. In V Arvind and R Ramanujam, editors, *18th Conference on Foundations of Software Technology and Theoretical Computer Science (Chennai, India, December 17–19, 1998)*, LNCS 1530. Springer-Verlag, December 1998.
7. D. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20:504–513, 1977.
8. Riccardo Focardi, Anna Ghelli, and Roberto Gorrieri. Using non interference for the analysis of security protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.
9. Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1), 1995.
10. Riccardo Focardi and Roberto Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23, 1997.
11. Riccardo Focardi and Roberto Gorrieri. Non interference: Past, present and future. In *Proceedings of DARPA Workshop on Foundations for Secure Mobile Code*, 1997.
12. C. Fournet, G. Gonthier, J.J. Levy, L. Marganget, and D. Remy. A calculus of mobile agents. In U. Montanari and V. Sassone, editors, *CONCUR: Proceedings of the International Conference on Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, August 1996. Springer-Verlag.
13. R. Reitmas G. Andrews. An axiomatic approach to information flow in programs. *ACM Transactions on Programming Languages and Systems*, 2(1):56–76, 1980.
14. J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and privacy*, 1992.
15. Nevin Heintz and Jon G. Riecke. The SLam calculus: Programming with secrecy and integrity. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998.

16. Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous pi-calculus. Technical report 2000:03, University of Sussex, 2000. Available from `http://www.cogs.susx.ac.uk/`.
17. Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In P. Wegner M. Tokoro, O. Nierstrasz, editor, *Proceedings of the ECOOP '91 Workshop on Object-Based Concurrent Computing*, volume 612 of *LNCS 612*. Springer-Verlag, 1992.
18. Kohei Honda, Vasco Vasconcelos, and Nobuko Yoshida Honda. Secure information flow as typed process behaviour. In *Proceedings of European Symposium on Programming (ESOP) 2000*. Springer-Verlag, 2000.
19. R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.
20. R. Milner, J. Parrow, and D. Walker. Mobile logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.
21. R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.
22. Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. Extended abstract in LICS '93.
23. Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University, 1997. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press.
24. James Riely and Matthew Hennessy. Resource access control in systems of mobile agents (extended abstract). In *Proceedings of 3rd International Workshop on High-Level Concurrent Languages*, Nice, France, September 1998. Full version available as Computer Science Technical Report 2/98, University of Sussex, 1997. Available from `http://www.cogs.susx.ac.uk/`.
25. A.W. Roscoe, J.C.P. Woodcock, and L. Wulf. Non-interference through determinism. In *European Symposium on Research in Computer Security*, volume 875 of *LNCS*, 1994.
26. P.Y.A. Ryan and S.A. Schneider. Process algebra and non-interference. In *CSFW 12*. IEEE, 1997.
27. Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *Conference Record of the ACM Symposium on Principles of Programming Languages*, San Diego, January 1998.

The Genomics Revolution and Its Challenges for Algorithmic Research

Richard M. Karp

University of California at Berkeley and
International Computer Science Institute
Berkeley, CA

Abstract. A fundamental goal of biology is to understand how living cells work. Recent developments in biotechnology and information processing have revolutionized this research field. Computational biology is a major component of this revolution and a fertile source of interesting problems related to algorithm design, combinatorics, statistics, combinatorial optimization, pattern recognition, data mining and computational learning theory. The speaker will provide an overview of this field, describing such areas as genomic mapping and sequencing, sequence analysis and analysis of gene expression data. He will then describe how his research in this field has called upon his background in theoretical computer science but required a shift in his approach to the design and development of algorithms.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, p. 428, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Alternating the Temporal Picture for Safety

Zohar Manna and Henny B. Sipma *

Computer Science Department
Stanford University
Stanford, CA. 94305-9045
{manna,sipma}@cs.stanford.edu

Abstract. We use alternating automata on infinite words to reduce the verification of linear temporal logic (LTL) safety properties over infinite-state systems to the proof of first-order verification conditions. This method generalizes the traditional deductive verification approach of providing verification rules for particular classes of formulas, such as invariances, nested precedence formulas, etc. It facilitates the deductive verification of arbitrary safety properties without the need for explicit temporal reasoning.

1 Introduction

Temporal logic is a powerful language for specifying properties of reactive systems. However, a specification language should be accompanied by verification methods to be useful in practice. For finite-state systems model checking provides such a verification method: it is (largely) automatic and applicable to arbitrary temporal properties. For infinite-state systems the situation is different. Although complete proof systems have been proposed for CTL [GF96] and LTL [MP91], the proof system presented for LTL requires the formula to be in a canonical form for the rules to be applicable. Transforming a formula into canonical form is expensive, the formula may grow exponentially, and worse, it may result in a formula that is so different from the original that the user's intuition proves useless for constructing the invariants and intermediate assertions necessary to complete the proof.

In this paper we present a verification rule SAFE that reduces the proof of LTL safety formulas to the proof of first-order validities. The rule SAFE constructs an *alternating automaton* [Var96,Var97] for the formula to be proven. This automaton may have to be strengthened by the user. First-order verification conditions are then generated based on the structure and labeling of this automaton.

The approach resembles that of *verification diagrams* [BMS95] and *assertion graphs* [BBM97] , which also reduce the proof of temporal properties to first-order

* This research was supported in part by the National Science Foundation under grant CCR-98-04100 and CCR-99-00984 ARO under grants DAAH04-96-1-0122 and DAAG55-98-1-0471, ARO under MURI grant DAAH04-96-1-0341, by Army contract DABT63-96-C-0096 (DARPA), and by Air Force contract F33615-99-C-3014.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 429-450, 2000
© Springer-Verlag Berlin Heidelberg 2000

verification conditions. In principle verification diagrams can be generated automatically (apart from the strengthening and refinement necessary for fairness), using an algorithm similar to the tableau construction for LTL, thus reducing a temporal property to first-order verification conditions. However, verification diagrams are based on (nondetermistic) ω-automata, and the size of the resulting diagram can, in the worst case, be exponential in the size of the formula, giving rise to a number of first-order verification conditions of the same order of magnitude, which clearly is undesirable. Alternating automata have an advantage over the regular ω-automata that they are linear in the size of the formula, thus making the number of verification conditions generated also proportional to it.

The remainder of the paper is organized as follows. Section 2 provides the preliminaries: it presents our computational model of transition systems, and specification language of linear temporal logic (LTL). Alternating automata and their models are introduced in Section 3. In Section 4 we give an algorithm to construct an alternating automaton for future LTL formulas, and we prove that the language accepted by the constructed automaton is precisely the set of sequences of states that satisfy the formula. Section 5 proposes the verification rules B-SAFE and SAFE that reduce the verification of future safety formulas to first-order verification conditions, and it is shown that the special verification rules of [MP95] are subsumed by this rule. In Section 6 we give an algorithm to construct an alternating automaton for LTL formulas involving past operators, and we propose a verification rule for such formulas. Finally, in Section 7 we discuss some of the limitations of these rules and give some ideas on how they could be overcome.

2 Preliminaries

2.1 Computational Model: Fair Transition Systems

The computational model used for reactive systems is that of a *transition system* [MP95] (TS), $\mathcal{S} = \langle V, \Theta_{\mathcal{S}}, \mathcal{T} \rangle$, where V is a finite set of variables, $\Theta_{\mathcal{S}}$ is an initial condition, and $\mathcal{T}$ is a finite set of transitions. A *state* s is an interpretation of V, and Σ denotes the set of all states. A transition $\tau \in \mathcal{T}$ is a function $\tau : \Sigma \mapsto 2^{\Sigma}$, and each state in $\tau(s)$ is called a τ-successor of s. We say that a transition τ is *enabled* on s if $\tau(s) \neq \emptyset$, otherwise τ is *disabled* on s. Each transition τ is represented by a *transition relation* $\rho_\tau(s, s')$, an assertion that expresses the relation between the values of V in s and the values of V (referred to by V') in any of its τ-successors s'.

A *run* of $\mathcal{S}$ is an infinite sequence of states such that the first state satisfies $\Theta_{\mathcal{S}}$ and any two consecutive states satisfy a ρ_τ for some $\tau \in \mathcal{T}$. A state s is called $\mathcal{S}$-*accessible* if it appears in some run of $\mathcal{S}$. The set of all runs of $\mathcal{S}$ is denoted by $\mathcal{L}(\mathcal{S})$.

2.2 Specification Language: Linear Temporal Logic

The specification language studied in this paper is *linear temporal logic*. We assume an underlying *assertion language* which is a first-order language over

interpreted symbols for expressing functions and relations over some concrete domains. We refer to a formula in the assertion language as a *state formula* or *assertion*. A *temporal formula* is constructed out of state formulas to which we apply the boolean connectives and the temporal operators shown below.

Temporal formulas are interpreted over a *model*, which is an infinite sequence of states $\sigma : s_0, s_1, \ldots$. Given a model σ, a state formula p and temporal formulas φ and ψ, we present an inductive definition for the notion of a formula φ holding at a position $j \geq 0$ in σ, denoted by $(\sigma, j) \models \varphi$.
For a state formula:

$$(\sigma, j) \models p \quad \text{iff} \quad s_j \models p, \text{ that is, } p \text{ holds on state } s_j.$$

For the boolean connectives:

$$\begin{array}{lll} (\sigma, j) \models \varphi \wedge \psi & \text{iff} & (\sigma, j) \models \varphi \text{ and } (\sigma, j) \models \psi \\ (\sigma, j) \models \varphi \vee \psi & \text{iff} & (\sigma, j) \models \varphi \text{ or } (\sigma, j) \models \psi \\ (\sigma, j) \models \neg\varphi & \text{iff} & (\sigma, j) \not\models \varphi \ . \end{array}$$

For the future temporal operators:

$$\begin{array}{lll} (\sigma, j) \models \bigcirc \varphi & \text{iff} & (\sigma, j+1) \models \varphi \\ (\sigma, j) \models \Box \varphi & \text{iff} & (\sigma, i) \models \varphi \text{ for all } i \geq j \\ (\sigma, j) \models \Diamond \varphi & \text{iff} & (\sigma, i) \models \varphi \text{ for some } i \geq j \\ (\sigma, j) \models \varphi \, \mathcal{U} \, \psi & \text{iff} & (\sigma, k) \models \psi \text{ for some } k \geq j, \\ & & \text{and } (\sigma, i) \models \varphi \text{ for every } i,\ j \leq i < k \\ (\sigma, j) \models \varphi \, \mathcal{W} \, \psi & \text{iff} & (\sigma, j) \models \varphi \, \mathcal{U} \, \psi \text{ or } (\sigma, j) \models \Box \varphi \ . \end{array}$$

For the past temporal operators:

$$\begin{array}{lll} (\sigma, j) \models \ominus \varphi & \text{iff} & j > 0 \text{ and } (\sigma, j-1) \models \varphi \\ (\sigma, j) \models \widetilde{\ominus} \varphi & \text{iff} & j = 0 \text{ or } (\sigma, j-1) \models \varphi \\ (\sigma, j) \models \boxminus \varphi & \text{iff} & (\sigma, i) \models \varphi \text{ for all } 0 \leq i \leq j \\ (\sigma, j) \models \diamondminus \varphi & \text{iff} & (\sigma, i) \models \varphi \text{ for some } 0 \leq i \leq j \\ (\sigma, j) \models \varphi \, \mathcal{S} \, \psi & \text{iff} & (\sigma, k) \models \psi \text{ for some } k \leq j, \\ & & \text{and } (\sigma, i) \models \varphi \text{ for every } i,\ k < i \leq j \\ (\sigma, j) \models \varphi \, \mathcal{B} \, \psi & \text{iff} & (\sigma, j) \models \varphi \, \mathcal{S} \, \psi \text{ or } (\sigma, j) \models \boxminus \varphi \ . \end{array}$$

An infinite sequence of states σ *satisfies* a temporal formula φ, written $\sigma \models \varphi$, if $(\sigma, 0) \models \varphi$. The set of all sequences that satisfy a formula φ is denoted by $\mathcal{L}(\varphi)$, the *language* of φ.

We say that a formula is a future (past) formula if it contains only state formulas, boolean connectives and future (past) temporal operators. We say that a formula is a general safety formula if it is of the form $\Box \varphi$, for a past formula φ.

A state formula p is called $\mathcal{S}$-*state valid* if it holds over all $\mathcal{S}$-accessible states. A temporal formula φ is called $\mathcal{S}$-*valid* (valid over system $\mathcal{S}$), denoted by

$$\mathcal{S} \models \varphi \ ,$$

if it holds over all runs of $\mathcal{S}$.

3 Alternating Automata

Alternating automata are a generalization of nondeterministic automata. Nondeterministic automata have an existential flavor: a word is accepted if it is accepted by *some* path through the automaton. On the other hand ∀-automata [MP87] have a universal flavor: a word is accepted if it is accepted by *all* paths. Alternating automata combine the two flavors by allowing choices along a path to be marked as either existential or universal.

Example Consider the two automata shown in Figure 1. An arc between two edges denotes an "*and*" choice: both paths have to be accepting. The absence of an arc denotes the (regular) "*or*" choice.

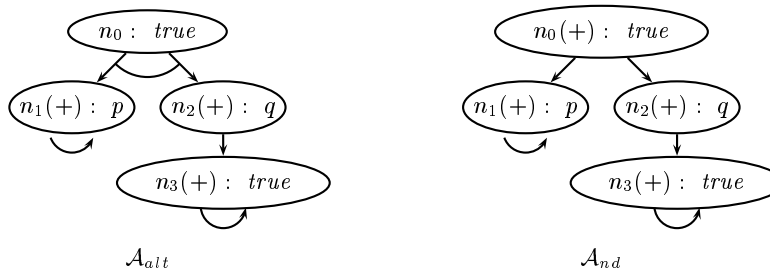


Fig. 1. An alternating automaton $\mathcal{A}_{alt}$ and a nondeterministic automaton $\mathcal{A}_{nd}$.

Automaton $\mathcal{A}_{alt}$ accepts only sequences of the form

$$\langle -, - \rangle, \langle p, q \rangle, \langle p, - \rangle, \langle p, - \rangle, \langle p, - \rangle, \ldots$$

where a "$-$" denotes a "don't care". For a sequence to be accepted, it has to be accepted by both branches. Automaton $\mathcal{A}_{nd}$, on the other hand, accepts sequences of the form

$$\langle -, - \rangle, \langle p, - \rangle, \langle p, - \rangle, \langle p, - \rangle, \langle p, - \rangle, \ldots$$

and sequences of the form

$$\langle -, - \rangle, \langle -, q \rangle, \langle -, - \rangle, \langle -, - \rangle, \langle -, - \rangle, \ldots$$

■

Automata are usually defined with input symbols labeling the edges. However, for our purposes it is more convenient to have them label the nodes. Therefore our definition of alternating automata is somewhat different from those found in [Var96,Var97].

Definition 1 (Alternating Automaton) An *alternating automaton* $\mathcal{A}$ is defined recursively as follows:

$$\begin{array}{lll} \mathcal{A} ::= \epsilon_{\mathcal{A}} & & \text{empty automaton} \\ \quad | \quad \langle \nu, \delta, f \rangle & & \text{single node} \\ \quad | \quad \mathcal{A} \wedge \mathcal{A} & & \text{conjunction of two automata} \\ \quad | \quad \mathcal{A} \vee \mathcal{A} & & \text{disjunction of two automata} \end{array}$$

where ν is a state formula, δ is an alternating automaton expressing the next-state relation, and f indicates whether the node is accepting (denoted by $+$) or rejecting (denoted by $-$). We require that the automaton be finite.

The set of nodes of an alternating automaton $\mathcal{A}$, denoted by $\mathcal{N}(\mathcal{A})$ is formally defined as

$$\begin{array}{lcl} \mathcal{N}(\epsilon_{\mathcal{A}}) & = & \emptyset \\ \mathcal{N}(\langle \nu, \delta, f \rangle) & = & \langle \nu, \delta, f \rangle \cup \mathcal{N}(\delta) \\ \mathcal{N}(\mathcal{A}_1 \wedge \mathcal{A}_2) & = & \mathcal{N}(\mathcal{A}_1) \cup \mathcal{N}(\mathcal{A}_2) \\ \mathcal{N}(\mathcal{A}_1 \vee \mathcal{A}_2) & = & \mathcal{N}(\mathcal{A}_1) \cup \mathcal{N}(\mathcal{A}_2) \end{array}$$

We denote with $\mathcal{N}_{rej}(\mathcal{A})$ the set of nodes of $\mathcal{A}$ that are rejecting, that is,

$$\mathcal{N}_{rej}(\mathcal{A}) = \{n \in \mathcal{N}(\mathcal{A}) \mid f(n) = -\} \ .$$

Example The automata shown in Figure 1 can be written as follows:

$$\mathcal{A}_{alt} : \langle \mathit{true}, \mathcal{A}_1 \wedge \langle q, \mathcal{A}_2, + \rangle, + \rangle \qquad \text{and} \qquad \mathcal{A}_{nd} : \langle \mathit{true}, \mathcal{A}_1 \vee \langle q, \mathcal{A}_2, + \rangle, + \rangle$$

where

$$\mathcal{A}_1 = \langle p, \mathcal{A}_1, + \rangle \qquad \text{and} \qquad \mathcal{A}_2 = \langle \mathit{true}, \mathcal{A}_2, + \rangle \ .$$

■

A path through a regular ω-automaton is an infinite sequence of nodes. A "path" through an alternating ω-automaton is, in general, a tree. To define the language of an alternating automaton, we first define a tree.

Definition 2 A *tree* is defined recursively as follows:

$$\begin{array}{lll} T ::= \epsilon_T & & \text{empty tree} \\ \quad | \quad T \cdot T & & \text{composition} \\ \quad | \quad \langle \mathit{node}, T \rangle & & \text{single node with child tree} \end{array}$$

A tree may have both finite and infinite branches.

Definition 3 (Run) Given an infinite sequence of states $\sigma : s_0, s_1, \ldots$, a tree T is called a *run* of σ in $\mathcal{A}$ if one of the following holds:

$$\begin{array}{lll} \mathcal{A} = \epsilon_{\mathcal{A}} & \text{and} & T = \epsilon_T \\ \mathcal{A} = n & \text{and} & T = \langle n, T' \rangle \text{ and } s_0 \models \nu(n) \text{ and} \\ & & T' \text{ is a run of } s_1, s_2, \ldots \text{ in } \delta(n) \\ \mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2 & \text{and} & T = T_1 \cdot T_2, \\ & & T_1 \text{ is a run of } \mathcal{A}_1 \text{ and } T_2 \text{ is a run of } \mathcal{A}_2 \\ \mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2 & \text{and} & T \text{ is a run of } \mathcal{A}_1 \text{ or } T \text{ is a run of } \mathcal{A}_2 \end{array}$$

Definition 4 (Accepting run) A run T is *accepting* if every infinite branch contains infinitely many accepting nodes.

Example A run of the sequence

$$\sigma : \langle p, q\rangle, \langle p, q\rangle, \langle p, \neg q\rangle, \langle p, q\rangle, \ldots$$

in the automaton $\mathcal{A}_{alt}$ of Figure 1 is shown in Figure 2. The run is clearly accepting since both branches contain infinitely many accepting nodes. ■

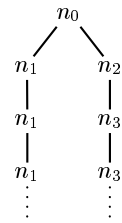


Fig. 2. Run of $\sigma : \langle p, q\rangle, \langle p, q\rangle, \langle p, \neg q\rangle, \langle p, q\rangle, \ldots$ in $\mathcal{A}_{alt}$

Definition 5 (Model) An infinite sequence of states σ is a *model* of an alternating automaton $\mathcal{A}$ if there exists an accepting run of σ in $\mathcal{A}$.

The set of models of an automaton $\mathcal{A}$, also called the *language of* $\mathcal{A}$, is denoted by $\mathcal{L}(\mathcal{A})$.

4 Linear Temporal Logic: Future Formulas

It has been shown that for every LTL formula φ there exists an alternating automaton $\mathcal{A}$ such that $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A})$ and the size of $\mathcal{A}$ is linear in the size of φ [Var97]. In [Var97] a construction method is given for such an automaton with propositions labeling the edges. Since we prefer to label the nodes with propositions (or, in our case, state formulas), we present a slightly different procedure. In the remainder of this paper we assume that all negations have been pushed in to the state level (a full set of rewrite rules to accomplish this is given in [MP95]), that is, no temporal operator is in the scope of a negation.

Given an LTL formula φ, an alternating automaton $\mathcal{A}(\varphi)$ is constructed, as follows.
For a state formula p:

$$\mathcal{A}(p) = \langle p, \epsilon_{\mathcal{A}}, +\rangle \ .$$

For temporal formulas φ and ψ:

$$\begin{array}{lcl}
\mathcal{A}(\varphi \wedge \psi) & = & \mathcal{A}(\varphi) \wedge \mathcal{A}(\psi) \\
\mathcal{A}(\varphi \vee \psi) & = & \mathcal{A}(\varphi) \vee \mathcal{A}(\psi) \\
\mathcal{A}(\bigcirc \varphi) & = & \langle \mathit{true}, \mathcal{A}(\varphi), + \rangle \\
\mathcal{A}(\Box \varphi) & = & \langle \mathit{true}, \mathcal{A}(\Box \varphi), + \rangle \wedge \mathcal{A}(\varphi) \\
\mathcal{A}(\Diamond \varphi) & = & \langle \mathit{true}, \mathcal{A}(\Diamond \varphi), - \rangle \vee \mathcal{A}(\varphi) \\
\mathcal{A}(\varphi \, \mathcal{U} \, \psi) & = & \mathcal{A}(\psi) \vee (\langle \mathit{true}, \mathcal{A}(\varphi \, \mathcal{U} \, \psi), - \rangle \wedge \mathcal{A}(\varphi)) \\
\mathcal{A}(\varphi \mathcal{W} \psi) & = & \mathcal{A}(\psi) \vee (\langle \mathit{true}, \mathcal{A}(\varphi \mathcal{W} \psi), + \rangle \wedge \mathcal{A}(\varphi))
\end{array}$$

The constructions are illustrated in Figure 3.

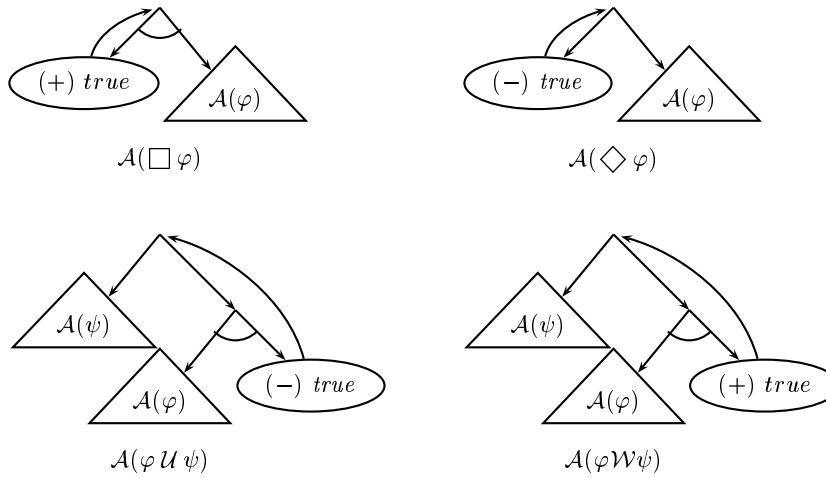


Fig. 3. Alternating automata for the temporal operators $\Box$, $\Diamond$, $\mathcal{U}$, $\mathcal{W}$

Note the close resemblance between these constructions and the expansion congruences [MP95]:

$$\begin{array}{lcl}
\Box \varphi & \approx & \varphi \wedge \bigcirc \Box \varphi \\
\Diamond \varphi & \approx & \varphi \vee \bigcirc \Diamond \varphi \\
\varphi \, \mathcal{U} \, \psi & \approx & \psi \vee (\varphi \wedge \bigcirc(\varphi \, \mathcal{U} \, \psi)) \\
\varphi \mathcal{W} \psi & \approx & \psi \vee (\varphi \wedge \bigcirc(\varphi \mathcal{W} \psi))
\end{array}$$

Alternating automata represent these congruences completely, while in addition capturing the acceptance condition, not encoded in them.

Example The automaton for the wait-for formula, for state formulas p, q, and r:

$$\varphi : \Box(p \rightarrow q \mathcal{W} r)$$

is shown in Figure 4. A run for the sequence

$$\sigma : \langle \neg p, -, - \rangle, \langle p, q, \neg r \rangle, \langle \neg p, q, \neg r \rangle, \langle p, \neg q, r \rangle, \langle \neg p, -, - \rangle, \langle \neg p, -, - \rangle, \ldots$$

is shown in Figure 5.

The automaton for the formula

$$\varphi : \Box(p \to q\,\mathcal{U}\,r)$$

is identical to that of Figure 4 except that node n_4 is rejecting. ■

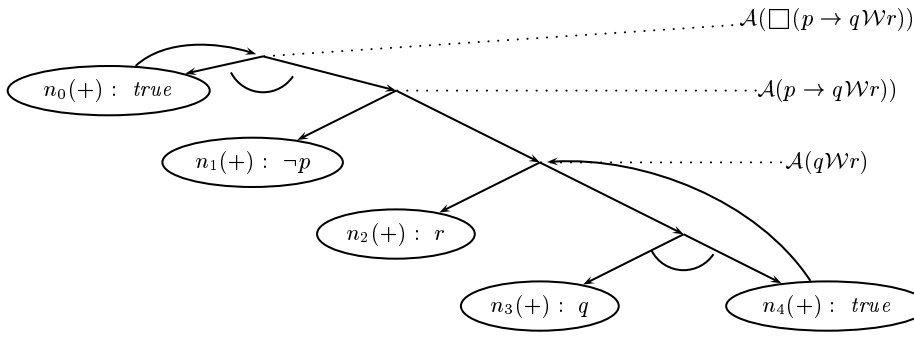


Fig. 4. Alternating automaton for $\Box(p \to q\mathcal{W}r)$

It is easy to see that, without changing the language of the automaton, we can make the following simplifications to the construction, for state formula p:

$$\begin{aligned}
\mathcal{A}(\Box\, p) &= \langle p, \mathcal{A}(\Box\, p), +\rangle \\
\mathcal{A}(p\,\mathcal{U}\,\psi) &= \mathcal{A}(\psi) \vee (\langle p, \mathcal{A}(p\,\mathcal{U}\,\psi), -\rangle \\
\mathcal{A}(p\mathcal{W}\psi) &= \mathcal{A}(\psi) \vee (\langle p, \mathcal{A}(p\mathcal{W}\psi), +\rangle
\end{aligned}$$

In this case the automaton for $\Box(p \to q\mathcal{W}r)$ becomes the one shown in Figure 6.

Theorem 1. *For a future temporal formula φ, $\mathcal{L}(\varphi) = \mathcal{L}(\mathcal{A}(\varphi))$.*

Before we prove this theorem, we state a supporting lemma, which we will use without mention in the proof below.

Lemma 1. *For two automata $\mathcal{A}_1$ and $\mathcal{A}_2$,*
(1) $\mathcal{L}(\mathcal{A}_1) \cap \mathcal{L}(\mathcal{A}_2) \;=\; \mathcal{L}(\mathcal{A}_1 \wedge \mathcal{A}_2)$
(2) $\mathcal{L}(\mathcal{A}_1) \cup \mathcal{L}(\mathcal{A}_2) \;=\; \mathcal{L}(\mathcal{A}_1 \vee \mathcal{A}_2)$

Proof This follows directly from the definition of a run. ■

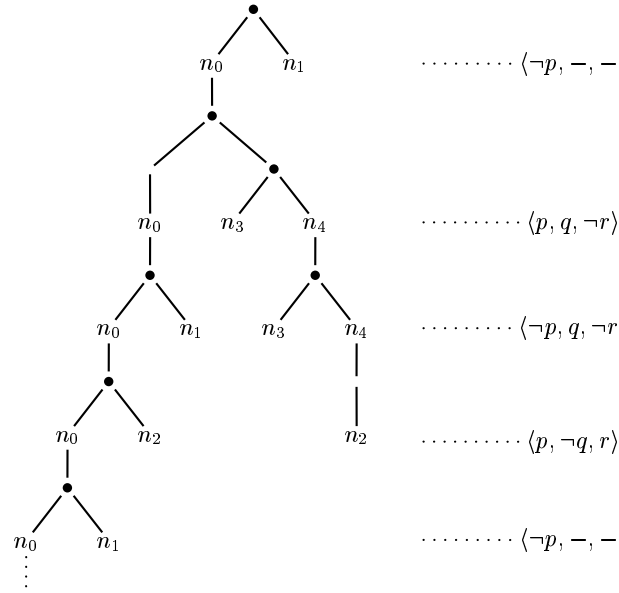


Fig. 5. Run

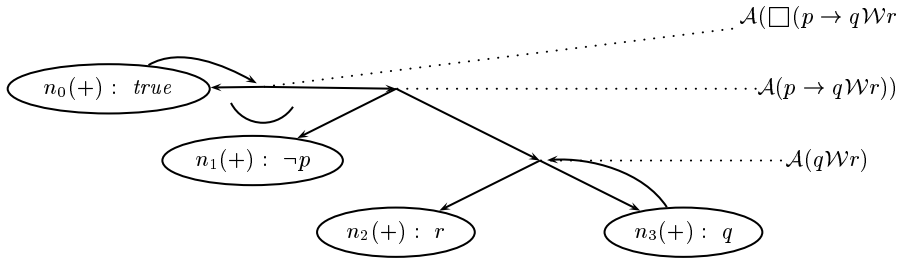


Fig. 6. (Simplified) Alternating automaton for $\Box(p \to q\mathcal{W}r)$

Proof of Theorem 1 The proof is by induction on the structure of the formula. In each of the cases we assume an arbitrary sequence of states $\sigma : s_0, s_1, \ldots$, and denote the sequence $s_i, s_{i+1}, \ldots$ by σ^i.

- φ is a state formula. In this case $\mathcal{A} = \langle \varphi, \epsilon_{\mathcal{A}}, + \rangle$. $\mathcal{A}$ accepts all sequences whose first state satisfies φ, which are exactly the sequences that satisfy φ.
- $\varphi = \varphi_1 \wedge \varphi_2$; $\varphi = \varphi_1 \vee \varphi_2$. These follow directly from Lemma 1.
- $\varphi = \bigcirc \psi$. In this case $\mathcal{A}(\varphi) = \langle \mathit{true}, \mathcal{A}(\psi), + \rangle$. We prove the two directions separately.
  - Assume $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$. Then σ has an accepting run T in $\mathcal{A}(\varphi)$ such that $T = \langle n, T' \rangle$, and T' is a run of σ^1 in $\mathcal{A}(\psi)$. Clearly if T is accepting, so is T', and thus $\sigma^1 \in \mathcal{L}(\mathcal{A}(\psi))$. Then, by the inductive hypothesis, $\sigma^1 \models \psi$, and thus, by the definition of $\bigcirc$, $\sigma \models \varphi$.
  - Assume $\sigma \models \bigcirc \psi$. Then, $\sigma^1 \models \psi$, and, by the inductive hypothesis, $\sigma^1 \in \mathcal{L}(\mathcal{A}(\psi))$, and, by the definition of a run, $\sigma \in \mathcal{L}(\langle \mathit{true}, \mathcal{A}(\psi), + \rangle)$.
- $\varphi = \Box \psi$. In this case $\mathcal{A}(\varphi) = \langle \mathit{true}, \mathcal{A}(\varphi), + \rangle \wedge \mathcal{A}(\psi)$. We prove the two directions separately.
  - Assume $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$, and for the sake of contradiction, $\sigma \not\models \Box \psi$. Then there must be some $i \geq 0$ such that $\sigma^i \not\models \psi$. But then, by the inductive hypothesis, $\sigma^i \notin \mathcal{L}(\mathcal{A}(\psi))$, and, by the definition of $\mathcal{A}(\Box \psi)$ (and by Lemma 1), $\sigma^i \notin \mathcal{L}(\mathcal{A}(\Box \psi))$. For $i = 0$ this contradicts the assumption that $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$. For $i > 0$ we have that $\sigma^{i-1} \notin \mathcal{L}(\langle \mathit{true}, \mathcal{A}(\Box \psi), + \rangle)$ and thus (again by Lemma 1 and the definition of $\mathcal{A}(\Box \psi)$) that $\sigma^{i-1} \notin \mathcal{L}(\mathcal{A}(\Box \psi))$. By downwards induction on i we can conclude that $\sigma \notin \mathcal{L}(\mathcal{A}(\Box \psi))$, a contradiction. Therefore $\sigma \models \Box \psi$.
  - Assume $\sigma \models \Box \psi$. Then, by the semantics of $\Box$, for all $i \geq 0$, $\sigma^i \models \psi$, and therefore, by the induction hypothesis, for all $i \geq 0$, $\sigma^i \in \mathcal{L}(\mathcal{A}(\psi))$. Let n_0 be $\langle \mathit{true}, \mathcal{A}(\Box \psi), + \rangle$. We construct a tree T as shown in Figure 7, where T_{σ^i} is an accepting run of σ^i in $\mathcal{A}(\psi)$, which exists since for all $i \geq 0$, $\sigma^i \in \mathcal{L}(\mathcal{A}(\psi))$. We claim that T is an accepting run of σ in $\mathcal{A}(\Box \psi)$ and therefore $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$. Indeed, consider an infinite branch in T. This branch is either $n_0, n_0, n_0, \ldots$ or it ends in a T_{σ^i}. In any case it contains an infinite number of accepting nodes.
- $\varphi = \Diamond \psi$. In this case $\mathcal{A}(\varphi) = \langle \mathit{true}, \mathcal{A}(\varphi), - \rangle \vee \mathcal{A}(\psi)$.
  - Assume $\sigma \in \mathcal{L}(\mathcal{A}(\varphi))$ and let T be a run of σ in $\mathcal{L}(\mathcal{A}(\varphi))$. Let n_0 be $\langle \mathit{true}, \mathcal{A}(\varphi), - \rangle$. Then T has to be of the form $n_0, \ldots, n_0, T'$, where T' is an accepting run of $\mathcal{A}(\psi)$. (Note that it cannot be $n_0, n_0, \ldots$, because n_0 is rejecting.) Therefore there exists a suffix σ^i of σ such that $\sigma^i \in \mathcal{L}(\mathcal{A}(\psi))$. By the induction hypothesis, $\sigma^i \models \psi$ and therefore $\sigma \models \Diamond \psi$.
  - Assume $\sigma \models \Diamond \psi$. Then, by the semantics of $\Diamond$, there exists $i \geq 0$ such that $\sigma^i \models \psi$, and by the inductive hypothesis, $\sigma^i \in \mathcal{L}(\mathcal{A}(\psi))$, and thus (by the definition of $\mathcal{A}(\Diamond \psi)$ and Lemma 1) $\sigma^i \in \mathcal{L}(\mathcal{A}(\Diamond \psi))$. If $i = 0$ we are done. If $i > 0$ then we have that $\sigma^{i-1} \in \mathcal{L}(\langle \mathit{true}, \mathcal{A}(\Diamond \psi), - \rangle)$, and thus $\sigma^{i-1} \in \mathcal{L}(\mathcal{A}(\Diamond \psi))$. By downwards induction on i we can conclude that $\sigma \in \mathcal{L}(\Diamond \psi)$.
- $\varphi = \varphi_1 \, \mathcal{U} \, \varphi_2$; $\varphi = \varphi_1 \mathcal{W} \varphi_2$. The proof of these cases proceeds along the same lines as those for $\Box \psi$ and $\Diamond \psi$, and is omitted.

■

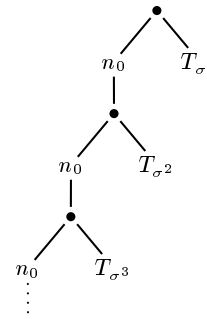


Fig. 7. Tree for σ in $\mathcal{A}(\Box\,\psi)$

5 Temporal Verification Rule for Future Safety Formulas

Alternating automata can be used to automatically reduce the verification of an arbitrary safety property specified by a future formula to first-order verification conditions, where a safety property is defined to be a property φ, such that if a sequence σ does not satisfy φ, then there is a finite prefix of σ such that φ is false on every extension of this prefix.

We define the *initial condition* of an alternating automaton $\mathcal{A}$, denoted by $\theta_{\mathcal{A}}(\mathcal{A})$, as follows:

$$\begin{array}{lcl}
\theta_{\mathcal{A}}(\epsilon_{\mathcal{A}}) & = & \mathit{true} \\
\theta_{\mathcal{A}}(\langle \nu, \delta, f \rangle) & = & \nu \\
\theta_{\mathcal{A}}(\mathcal{A}_1 \wedge \mathcal{A}_2) & = & \theta_{\mathcal{A}}(\mathcal{A}_1) \wedge \theta_{\mathcal{A}}(\mathcal{A}_2) \\
\theta_{\mathcal{A}}(\mathcal{A}_1 \vee \mathcal{A}_2) & = & \theta_{\mathcal{A}}(\mathcal{A}_1) \vee \theta_{\mathcal{A}}(\mathcal{A}_2)
\end{array}$$

Intuitively, the initial condition of an automaton characterizes the set of initial states of sequences accepted by the automaton. For example, the initial condition of the automaton shown in Figure 4 is

$$\theta_{\mathcal{A}}(\mathcal{A}) = \neg p \vee q \vee r \ .$$

Basic Rule

Following the style of verification rules of [MP95] we can now present the basic temporal rule B-SAFE, shown in Figure 8. In the rule we use the *Hoare triple* notation $\{p\}\,\tau\,\{q\}$, which stands for $p \wedge \rho_\tau \rightarrow q'$. The notation $\{p\}\,\mathcal{T}\,\{q\}$ stands for $\{p\}\,\tau\,\{q\}$ for all $\tau \in \mathcal{T}$.

Premise T1, the *Initiation Condition*, requires that the initial condition of $\mathcal{S}$ implies the initial condition of the automaton $\mathcal{A}(\varphi)$. Premise T2, the *Consecution Condition*, requires that for all nodes, $n \in \mathcal{N}(\mathcal{A}(\varphi))$, and for all transitions $\tau \in \mathcal{T}$, τ, if enabled, leads to the initial condition of the next-state automaton of n.

For a future safety formula φ and TS $\mathcal{S} : \langle V, \Theta_{\mathcal{S}}, \mathcal{T} \rangle$,

T1. $\Theta_{\mathcal{S}} \rightarrow \theta_{\mathcal{A}}(\mathcal{A}(\varphi))$

T2. $\{\nu(n)\}\, \mathcal{T}\, \{\theta_{\mathcal{A}}(\delta(n))\}$ for $n \in \mathcal{N}(\mathcal{A}(\varphi))$

$\mathcal{S} \models \varphi$

Fig. 8. Basic temporal rule B-SAFE

General Rule

As is the case with the rules B-INV and B-WAIT in [MP95], rule B-SAFE is hardly ever directly applicable, because the assertions labeling the nodes are not inductive: they must be strengthened. To represent the strengthening of an automaton, we add a new label μ to the definition of a node, $\langle \mu, \nu, \delta, f \rangle$, where μ is an assertion, and we change the definition of $\theta_{\mathcal{A}}$ for a node into

$$\theta_{\mathcal{A}}(\langle \mu, \nu, \delta, f \rangle) = \mu \ .$$

Using these definitions, Figure 9 shows the more general rule SAFE that allows strengthening of the intermediate assertions.

For a future safety formula φ, TS $\mathcal{S} : \langle V, \Theta_{\mathcal{S}}, \mathcal{T} \rangle$,
and strengthened automaton $\mathcal{A}(\varphi)$

T0. $\mu(n) \rightarrow \nu(n)$ for $n \in \mathcal{N}(\mathcal{A}(\varphi))$

T1. $\Theta_{\mathcal{S}} \rightarrow \theta_{\mathcal{A}}(\mathcal{A}(\varphi))$

T2. $\{\mu(n)\}\, \mathcal{T}\, \{\theta_{\mathcal{A}}(\delta(n))\}$ for $n \in \mathcal{N}(\mathcal{A}(\varphi))$

$\mathcal{S} \models \varphi$

Fig. 9. General temporal rule SAFE

Note that terminal nodes, that is, nodes with $\delta = \epsilon_{\mathcal{A}}$, never need to be strengthened. This is so, because consecution conditions from terminal nodes are all of the form $\mu(n) \wedge \rho_\tau \rightarrow \mathit{true}$, since $\theta_{\mathcal{A}}(\epsilon_{\mathcal{A}}) = \mathit{true}$, and thus trivially valid.

Some strengthening can be applied automatically. For automata of the form

$$\mathcal{A} : \ n_0 : \langle \mathit{true}, \mathcal{A}, f \rangle \wedge \mathcal{A}_1$$

node n_0 can be strengthened with $\theta_{\mathcal{A}}(\mathcal{A}_1)$ without changing the language of the automaton. In the following special cases we will apply this default strengthening.

Special Cases

We consider the relation between rule SAFE and the special verification rules of [MP95].

INV

For an invariance formula $\varphi : \Box p$, with p a state formula, the (simplified) automaton $\mathcal{A}(\varphi)$ consists of a single node n_0, labeled by p and with next-state automaton n_0. If we strengthen node n_0 with χ, rule SAFE produces the following verification conditions (after applying the default strengthening):

$$\begin{array}{lll} \text{T0.} & \chi \rightarrow p & \text{for } n_0 \\ \text{T1.} & \Theta_{\mathcal{S}} \rightarrow \chi & \\ \text{T2.} & \{\chi\}\,\mathcal{T}\,\{\chi\} & \text{for } n_0 \end{array}$$

which is identical to rule INV.

WAIT

For a wait-for formula,

$$\varphi : \ \Box(p \rightarrow q\mathcal{W}r)$$

rule SAFE, using the automaton shown in Figure 6 with node n_3 strengthened by χ, results in the following verification conditions (after applying the default strengthening):

$$\begin{array}{lll} \text{T0.} & \chi \rightarrow q & \text{for } n_3 \\ \text{T1.} & \Theta_{\mathcal{S}} \rightarrow \neg p \vee r \vee \chi & \\ \text{T2.} & \{\neg p \vee r \vee \chi\}\,\mathcal{T}\,\{\neg p \vee r \vee \chi\} & \text{for } n_0 \\ & \{\chi\}\,\mathcal{T}\,\{\chi \vee r\} & \text{for } n_3 \end{array}$$

Comparing these conditions with those of rule WAIT we notice that the state-validity $p \rightarrow \chi \vee r$ in WAIT has been replaced by the invariance conditions for the same formula, represented by T1 and the first set of conditions of T2. The other conditions are identical.

N-WAIT

Consider the nested wait-for formula, with p, $q_0, \ldots, q_n$ state formulas:

$$\Box(p \to q_n \mathcal{W}(q_{n-1} \ldots \mathcal{W}(q_1 \mathcal{W} q_0) \ldots))$$

The procedure described above generates the (simplified) automaton shown in Figure 10 for this formula. It is easy to see that this automaton will generate the same verification conditions as given in NWAIT, again with the exception that the state validity in NWAIT is replaced by the invariance conditions for the same formula.

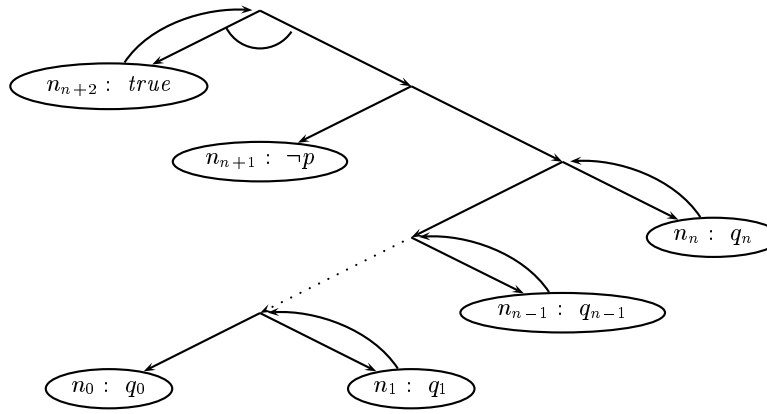


Fig. 10. Alternating automaton for $\Box(p \to q_n \mathcal{W}(q_{n-1} \ldots \mathcal{W}(q_1 \mathcal{W} q_0) \ldots))$

Theorem 2 (Soundness of B-SAFE**).** *For a* TS $\mathcal{S}$ *and future safety formula* φ*, if the premises T1 and T2 of rule* B-SAFE *are* $\mathcal{S}$*-state valid then* $\mathcal{S} \models \varphi$*.*

Before we prove this theorem we present some supporting definitions and lemmas.

Definition 6 (k-Run) A finite tree T is a *k-run* of an infinite sequence of states $\sigma : s_0, s_1, \ldots$ in an alternating automaton $\mathcal{A}$ if one of the following holds:

$\mathcal{A} = \epsilon_{\mathcal{A}}$	and	$T = \epsilon_T$
$\mathcal{A} = n$	and	$T = \langle n, T' \rangle$ and $s_0 \models \nu(n)$ and
		(a) $k = 1$ and $T' = \epsilon_T$, or
		(b) $k > 1$ and T' is a $(k-1)$-run of σ^1 in $\delta(n)$
$\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$	and	$T = T_1 \cdot T_2$ and T_1 is a k-run of σ in $\mathcal{A}_1$
		and T_2 is a k-run of σ in $\mathcal{A}_2$
$\mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2$	and	T is a k-run of σ in $\mathcal{A}_1$ or T is a k-run of σ in $\mathcal{A}_2$

Definition 7 (k-Model) An infinite sequence of states σ is a k-model of an alternating automaton $\mathcal{A}$ if there exists a k-run of σ in $\mathcal{A}$.

Example The sequence

$$\langle \neg p, -, - \rangle, \langle p, q, \neg r \rangle, \langle -, -, - \rangle, \ldots$$

is a 2-model of the automaton shown in Figure 4. Its 2-run is shown in Figure 11. ■

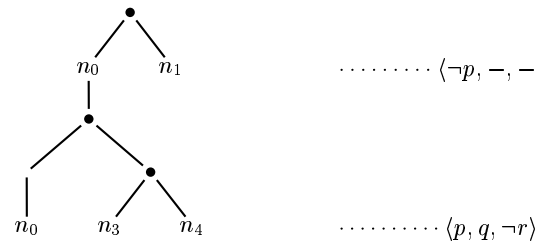


Fig. 11. 2-run of $\langle \neg p, -, - \rangle, \langle p, q, \neg r \rangle, \langle -, -, - \rangle, \ldots$

Definition 8 (k-Nodes) The set of k-nodes of a tree T, written $\mathcal{N}_k(T)$ consists of the nodes present at depth k. Formally:

$$\mathcal{N}_k(\epsilon_T) = \emptyset$$

$$\mathcal{N}_k(\langle n, T \rangle) = \begin{cases} \{n\} & \text{if } k = 1 \\ \mathcal{N}_{k-1}(T) & \text{otherwise} \end{cases}$$

$$\mathcal{N}_k(T_1 \cdot T_2) = \mathcal{N}_k(T_1) \cup \mathcal{N}_k(T_2)$$

Example For the tree T shown in Figure 5, $\mathcal{N}_1(T) = \{n_0, n_1\}$, $\mathcal{N}_2(T) = \{n_0, n_3, n_4\}$, $\mathcal{N}_3(T) = \{n_0, n_1, n_3, n_4\}$, etc. ■

Lemma 2. *For an infinite sequence of states $\sigma : s_0, s_1, \ldots$, $k > 0$, and alternating automata $\mathcal{A}$, $\mathcal{A}_1$, and $\mathcal{A}_2$ the following hold:*

- *σ is a k-model of $\epsilon_{\mathcal{A}}$.*
- *σ is a k-model of $\mathcal{A} = n$ iff $s_0 \models \nu(n)$ and σ^1 is a $(k-1)$-model of $\delta(n)$.*
- *σ is a k-model of $\mathcal{A}_1 \wedge \mathcal{A}_2$ iff σ is a k-model of $\mathcal{A}_1$ and σ is a k-model of $\mathcal{A}_2$.*
- *σ is a k-model of $\mathcal{A}_1 \vee \mathcal{A}_2$ iff σ is a k-model of $\mathcal{A}_1$ or σ is a k-model of $\mathcal{A}_2$.*

Proof These follow directly from the definition of k-run and k-model. ■

Lemma 3 (Initial Condition). *An infinite sequence of states $\sigma : s_0, \ldots$ is a 1-model of an alternating automaton $\mathcal{A}$ iff $s_0 \models \theta_{\mathcal{A}}(\mathcal{A})$.*

Proof The proof is by induction on the structure of the automaton.

- $\mathcal{A} = \epsilon_{\mathcal{A}}$. By lemma 2, the sequence σ is a 1-model of $\mathcal{A}$; $\theta_{\mathcal{A}}(\epsilon_{\mathcal{A}}) = \mathit{true}$ and thus $s_0 \models \theta_{\mathcal{A}}(\mathcal{A})$.
- $\mathcal{A} = n$. The sequence σ is a 1-model of $\mathcal{A}$ iff $s_0 \models \nu(n)$ iff $s_0 \models \theta_{\mathcal{A}}(\mathcal{A})$.
- $\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2$. By lemma 2, the sequence σ is a 1-model of $\mathcal{A}$ iff σ is a 1-model of $\mathcal{A}_1$ and σ is a 1-model of $\mathcal{A}_2$, iff, by the inductive hypothesis, $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}_1)$ and $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}_2)$, iff $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}_1) \wedge \theta_{\mathcal{A}}(\mathcal{A}_2)$ iff, by the definition of $\theta_{\mathcal{A}}$, $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}_1 \wedge \mathcal{A}_2)$.
- $\mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2$. Similar to the above case.

■

Lemma 4. *If an infinite sequence of states σ is a k-model of an alternating automaton $\mathcal{A}$ with k-run T, and if for all nodes $n \in \mathcal{N}_k(T)$, σ^k is a 1-model of $\delta(n)$, then σ is a $(k+1)$-model of $\mathcal{A}$.*

Proof Assume T is a k-run of σ in $\mathcal{A}$, and for all $n \in \mathcal{N}_k(T)$ σ^k is a 1-model of $\delta(n)$ with 1-run T_n. Then we can construct a tree T' by replacing each occurrence of $\langle n, \epsilon_T \rangle$ in T by $\langle n, T_n \rangle$. It is easy to see that T' is a $(k+1)$-run of σ in $\mathcal{A}$, and thus σ is a $(k+1)$-model of $\mathcal{A}$. ■

Lemma 5. *An infinite sequence of states $\sigma : s_0, s_1 \ldots$ is a k-model, with $k > 0$, of an alternating automaton $\mathcal{A}$, with k-run T iff for all $1 \leq j \leq k$, for all nodes $n \in \mathcal{N}_j(T)$, $s_{j-1} \models \nu(n)$.*

Proof By induction on k, and the structure of the tree T. ■

Example Consider Figure 5. The state $s_2 : \langle \neg p, q, \neg r \rangle$ indeed satisfies the assertion ν of all nodes in $\mathcal{N}_3(T) = \{n_0, n_1, n_3, n_4\}$. ■

Lemma 6. *Given a* TS *$\mathcal{S}$, a future temporal formula φ, and an infinite sequence of states $\sigma \in \mathcal{L}(\mathcal{S})$, if the premises T1 and T2 are $\mathcal{S}$-state valid, then σ is a k-model of $\mathcal{A}(\varphi)$ for all $k > 0$.*

Proof The proof is by induction on k.

- Base case. By the definition of a run of $\mathcal{S}$, $s_0 \models \Theta_{\mathcal{S}}$, and thus, by premise T1, $s_0 \models \theta_{\mathcal{A}}(\mathcal{A}(\varphi))$. Then, by lemma 3, σ is a 1-model of $\mathcal{A}(\varphi)$.
- Inductive step. Assume σ is a k-model of $\mathcal{A}(\varphi)$ with k-run T. Then, by lemma 5, for all $n \in \mathcal{N}_k(T)$, $s_{k-1} \models \nu(n)$. By the consecution requirement for runs, there exists some transition $\tau \in \mathcal{T}$ such that (s_{k-1}, s_k) satisfies ρ_τ. By premise T2, for every node $n \in \mathcal{N}(\mathcal{A}(\varphi))$, every transition starting from a state that satisfies $\nu(n)$ leads to a state that satisfies $\theta_{\mathcal{A}}(\delta(n))$, and thus, by lemma 3 and lemma 4, σ is a $(k+1)$-model of $\mathcal{A}(\varphi)$.

■

Lemma 7. *For a safety formula φ, if a sequence of states σ is a k-model of $\mathcal{A}(\varphi)$ for any $k > 0$, then σ is a model of $\mathcal{A}(\varphi)$.*

Proof This follows directly from the definition of a safety formula: a safety property holds iff it cannot be violated in finite time. ■

Proof of Theorem 2 Consider a TS $\mathcal{S}$ and a future safety formula φ and assume that the premises T1 and T2 are $\mathcal{S}$-state valid. Consider an arbitrary sequence of states $\sigma \in \mathcal{L}(\mathcal{S})$, and future safety formula φ. By lemma 6, σ is a k-model of $\mathcal{A}(\varphi)$ for all $k > 0$. Then, by lemma 7, σ is a model of $\mathcal{A}(\varphi)$. Finally, by theorem 1, $\sigma \models \varphi$. ■

6 Past formulas

In Section 4 we showed how to construct alternating automata for LTL formulas containing future operators only. Here we will extend the procedure to include past operators as well.

To define an alternating automaton for LTL formulas including past operators, we add a component g to the definition of a node, such that a node is now defined as

$$\langle \nu, \delta, f, g \rangle$$

where g indicates whether the node is past (indicated by "$\leftarrow$") or future (indicated by "$\rightarrow$").

To accomodate the presence of past nodes we extend the notions of a run and model of an automaton.

Definition 9 (Run) Given an infinite sequence of states $\sigma : s_0, s_1, \ldots$, and a position $j \geq 0$, a tree T is called a run of σ at position j if one of the following holds:

$$\begin{array}{lll}
\mathcal{A} = \epsilon_{\mathcal{A}} & \text{and} & T = \epsilon_T \\
\mathcal{A} = n & \text{and} & T = \langle n, T' \rangle \text{ and } s_0 \models \nu(n) \text{ and} \\
 & & \left\{ \begin{array}{l} \text{(a) } T' \text{ is a run of } \sigma \text{ in } \delta(n) \text{ at } j+1, \\ \quad \text{if } g(n) = \rightarrow, \text{ or} \\ \text{(b) } T' \text{ is a run of } \sigma \text{ in } \delta(n) \text{ at } j-1, \\ \quad \text{if } g(n) = \leftarrow \text{ and } j > 0, \text{ or} \\ \text{(c) } T' = \epsilon_T \text{ if } g(n) = \leftarrow, f(n) = +, \text{ and } j = 0. \end{array} \right. \\
\mathcal{A} = \mathcal{A}_1 \wedge \mathcal{A}_2 & \text{and} & T = T_1 \cdot T_2, \\
 & & T_1 \text{ is a run of } \mathcal{A}_1 \text{ and } T_2 \text{ is a run of } \mathcal{A}_2 \\
\mathcal{A} = \mathcal{A}_1 \vee \mathcal{A}_2 & \text{and} & T \text{ is a run of } \mathcal{A}_1 \text{ or } T \text{ is a run of } \mathcal{A}_2
\end{array}$$

Definition 10 (Accepting run) A run T is *accepting* if every infinite branch of T contains infinitely many accepting nodes.

Definition 11 (Model at j) An infinite sequence of states σ is a model of an alternating automaton $\mathcal{A}$ at position j if there exists an accepting run of σ in $\mathcal{A}$ at position j.

Definition 12 (Model) An infinite sequence of states σ is a model of an alternating automaton $\mathcal{A}$ if it is a model at position 0.

Given an LTL formula φ, an alternating automaton $\mathcal{A}(\varphi)$ is constructed as before, where all nodes constructed before are future nodes, and with the following additions for the past operators:

$$\begin{array}{lcl}
\mathcal{A}(\circledcirc\,\varphi) & = & \langle \mathit{true}, \mathcal{A}(\varphi), +, \leftarrow\rangle \\
\mathcal{A}(\ominus\,\varphi) & = & \langle \mathit{true}, \mathcal{A}(\varphi), -, \leftarrow\rangle \\
\mathcal{A}(\boxminus\,\varphi) & = & \langle \mathit{true}, \mathcal{A}(\boxminus\,\varphi), +, \leftarrow\rangle \wedge \mathcal{A}(\varphi) \\
\mathcal{A}(\diamondminus\,\varphi) & = & \langle \mathit{true}, \mathcal{A}(\diamondminus\,\varphi), -, \leftarrow\rangle \vee \mathcal{A}(\varphi) \\
\mathcal{A}(\varphi\,\mathcal{S}\,\psi) & = & \mathcal{A}(\psi) \vee (\langle \mathit{true}, \mathcal{A}(\varphi\,\mathcal{S}\,\psi), -, \leftarrow\rangle \wedge \mathcal{A}(\varphi)) \\
\mathcal{A}(\varphi\,\mathcal{B}\,\psi) & = & \mathcal{A}(\psi) \vee (\langle \mathit{true}, \mathcal{A}(\varphi\,\mathcal{B}\,\psi), +, \leftarrow\rangle \wedge \mathcal{A}(\varphi))
\end{array}$$

Again, these constructions closely resemble the expansion congruences for the past formulas:

$$\begin{array}{lcl}
\boxminus\,\varphi & \approx & \varphi \wedge \circledcirc\,\boxminus\,\varphi \\
\diamondminus\,\varphi & \approx & \varphi \vee \ominus\,\diamondminus\,\varphi \\
\varphi\,\mathcal{S}\,\psi & \approx & \psi \vee (\varphi \wedge \ominus(\varphi\,\mathcal{S}\,\psi)) \\
\varphi\,\mathcal{B}\,\psi & \approx & \psi \vee (\varphi \wedge \circledcirc\,(\varphi\,\mathcal{B}\,\psi))
\end{array}$$

Example For a causality formula φ : $\Box(p \rightarrow \diamondminus\, r)$ with p, q, and r state formulas, the automaton is shown in Figure 12. ■

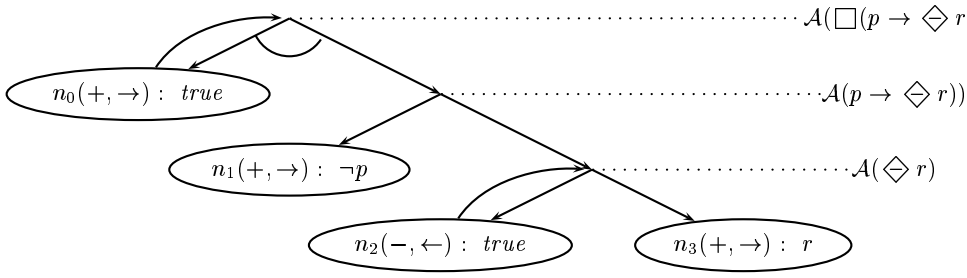


Fig. 12. Alternating automaton for $\Box(p \rightarrow \diamondminus\, r)$

We denote with $\mathcal{PN}(\mathcal{A})$ the past nodes of $\mathcal{A}$ and with $\mathcal{FN}(\mathcal{A})$ the future nodes of $\mathcal{A}$.

We can now formulate the verification rule GSAFE, shown in Figure 13, that is applicable to arbitrary general safety formulas. Again, we augment the definition of a node with a strengthening μ.

For a general safety formula φ, and TS $\mathcal{S} : \langle V, \Theta_{\mathcal{S}}, \mathcal{T} \rangle$,
and strengthened automaton $\mathcal{A}(\varphi)$

T0.	$\mu(n) \rightarrow \nu(n)$	for $n \in \mathcal{N}(\mathcal{A}(\varphi))$
T1.	$\Theta_{\mathcal{S}} \rightarrow \theta_{\mathcal{A}}(\mathcal{A}(\varphi))$	
T2.	$\{\mu(n)\}\, \mathcal{T}\, \{\theta_{\mathcal{A}}(\delta(n))\}$	for $n \in \mathcal{FN}(\mathcal{A}(\varphi))$
	$\{\mu(n)\}\, \mathcal{T}^{-1}\, \{\theta_{\mathcal{A}}(\delta(n))\}$	for $n \in \mathcal{PN}(\mathcal{A}(\varphi))$
T3.	$\Theta_{\mathcal{S}} \rightarrow \neg\mu(n)$	for $n \in \mathcal{PN}_{rej}(\mathcal{A}(\varphi))$
	$\mathcal{S} \models \varphi$	

Fig. 13. General safety rule GSAFE

Premise T0 requires that the assertions used to strengthen the node imply the original assertions. Premise T1 requires that the initial condition of the system implies the (strengthened) initial condition of the automaton for φ. Premise T2 requires the regular consecution condition for all future nodes, and the inverse consecution condition for all past nodes, where $\mathcal{T}^{-1} = \{\tau^{-1} \mid \tau \in \mathcal{T}\}$ and

$$\{p\}\, \tau^{-1}\, \{q\} \;=\; p' \wedge \rho_\tau \rightarrow q \;.$$

Finally, premise T3 requires that no initial state satisfies a rejecting past node. This last requirement ensures that "promises for the past" are fulfilled before the first state is reached.

Special cases

As for future formulas we compare the verification conditions produced by rule GSAFE with the premises of a special verification rule involving past operators presented in [MP95].

CAUS

For a causality formula $\varphi :\ \Box(p \rightarrow \diamondminus\, r)$ with p, q, and r state formulas, rule GSAFE, based on the automaton shown in Figure 12 with node n_2 strengthened

with χ, results in the following verification conditions:

T0.	$\chi \rightarrow \mathit{true}$	for n_2
T1.	$\Theta_{\mathcal{S}} \rightarrow \neg p \vee \chi \vee r$	
T2.	$\{\neg p \vee \chi \vee r\}\, \mathcal{T}\, \{\neg p \vee \chi \vee r\}$	for n_0
	$\{\chi\}\, \mathcal{T}^{-1}\, \{\chi \vee r\}$	for n_2
T3.	$\Theta_{\mathcal{S}} \rightarrow \neg \chi$	

For state formulas p and r, the premises of the rule CAUS are the following:

C1.	$p \rightarrow \chi \vee r$
C2.	$\Theta_{\mathcal{S}} \rightarrow \neg \chi \vee r$
C3.	$\{\chi\}\, \mathcal{T}^{-1}\, \{\chi \vee r\}$

Premise C1 corresponds to premises T1, and T2 for n_0. Premise C2 is represented by the (stronger) premise T3, and premise C3 is identical to premise T2 for n_2.

7 Discussion

The work presented in this paper is a first attempt to use alternating automata as a basis for the deductive verification of LTL properties. We have shown that it successfully generalizes several of the special verification rules for the corresponding classes of formulas, thus obviating the need to implement these rules separately in a verification tool. Instead the single rule GSAFE suffices. The rule SAFE has been implemented in STeP, the Stanford Temporal Prover, a verification tool for algorithmic and deductive verification of reactive systems [BBC+95,BBC+00].

It is straightforward to extend rule SAFE to general reactivity properties by adding a fourth premise

$$\text{T3.}\ \ \Box \Diamond \neg \mu(n) \qquad \text{for all rejecting nodes } n \in \mathcal{N}_{rej}(\mathcal{A}(\varphi))$$

which can be reduced to first-order verification conditions by one of the rules given in [MP91]. The reason we have omitted this extension from this paper is that we found this rule not useful for many of these properties. We are currently working on a more general rule that allows the application of different proof methods on different parts of the automaton. This will be presented in a forthcoming paper.

8 Related Work

Alternating finite state automata over finite words were first introduced in [CKS81] and shown to be exponentially more succinct than nondeterministic

automata. In [MH84] it was shown that alternating finite automata on infinite words with Büchi acceptance conditions are as expressive as nondeterministic ω-automata with Büchi acceptance conditions, and in [MSS88,Var94] it was shown that for every LTL formula an alternating Büchi automaton can be constructed that is linear in the size of the formula. In [Var95,Var96,Var97] alternating automata are used as the basis for a model checking algorithm for finite-state systems and both linear and branching time temporal logics. In [Var98] a theory of two-way alternating automata on infinite trees is developed to check satisfiability of μ-calculus formulas with both forward and backward modalities. In that paper a direction is introduced in the next-state relation, redirecting a run to the parent node when encountering a past operator. The procedure presented in this paper takes the opposite approach for past operators: it reverses the direction in the sequence, instead of in the automaton.

Deductive verification methods using automata include a sound and complete proofrule based on $\forall$-automata [MP87], and generalized verification diagrams [BMS95,MBSU98].

Acknowledgements

We thank Anca Browne, Michael Colón, Bernd Finkbeiner, Matteo Slanina, Calogero Zarba, and Zhang Ting for their suggestions and comments on this paper.

References

[BBC+95] N.S. Bjørner, A. Browne, E.S. Chang, M. Colón, A. Kapur, Z. Manna, H.B. Sipma, and T.E. Uribe. STeP: The Stanford Temporal Prover, User's Manual. Technical Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University, November 1995. available from `http://www-step.stanford.edu/`.

[BBC+00] N.S. Bjørner, A. Browne, M. Colón, B. Finkbeiner, Z. Manna, H.B. Sipma, and T.E. Uribe. Verifying temporal properties of reactive systems: A STeP tutorial. *Formal Methods in System Design*, 16(3):227–270, June 2000.

[BBM97] N.S. Bjørner, A. Browne, and Z. Manna. Automatic generation of invariants and intermediate assertions. *Theoretical Computer Science*, 173(1):49–87, February 1997. Preliminary version appeared in 1st *Intl. Conf. on Principles and Practice of Constraint Programming*, vol. 976 of LNCS, pp. 589–623, Springer-Verlag, 1995.

[BMS95] A. Browne, Z. Manna, and H.B. Sipma. Generalized temporal verification diagrams. In *15th Conference on the Foundations of Software Technology and Theoretical Computer Science*, vol. 1026 of *Lecture Notes in Computer Science*, pages 484–498. Springer-Verlag, 1995.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *J. ACM*, 28:114–133, 1981.

[GF96] O. Grumberg and L. Fix. Verification of temporal properties. *Logic and Computation*, 6(3):343–361, June 1996.

[MBSU98] Z. Manna, A. Browne, H.B. Sipma, and T.E. Uribe. Visual abstractions for temporal verification. In A. Haeberer, editor, *Algebraic Methodology and Software Technology (AMAST'98)*, vol. 1548 of *Lecture Notes in Computer Science*, pages 28–41. Springer-Verlag, December 1998.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32, 1984.

[MP87] Z. Manna and A. Pnueli. Specification and verification of concurrent programs by $\forall$-automata. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, number 398 in Lecture Notes in Computer Science, pages 124–164. Springer-Verlag, Berlin, 1987. Also in *Proc. 14th ACM Symp. Princ. of Prog. Lang.*, Munich, Germany, pp. 1–12, January 1987.

[MP91] Z. Manna and A. Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):97–130, 1991.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.

[MSS88] D.E. Muller, A. Saoudi, and P.E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proc. 3rd IEEE Symp. Logic in Comp. Sci.*, pages 422–427, 1988.

[Var94] M.Y. Vardi. Nontraditional applications of automata theory. In M. Hagiya and J.C. Mitchell, editors, *Proc. International Symposium on Theoretical Aspects of Computer Software*, vol. 789 of *Lecture Notes in Computer Science*, pages 575–597. Springer-Verlag, 1994.

[Var95] M.Y. Vardi. Alternating automata and program verification. In J. van Leeuwen, editor, *Computer Science Today. Recent Trends and Developments*, vol. 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer-Verlag, 1995.

[Var96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency. Structure versus Automata*, vol. 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, 1996.

[Var97] M.Y. Vardi. Alternating automata: Checking truth and validity for temporal logics. In *Proc. of the* 14th *Intl. Conference on Automated Deduction*, vol. 1249 of *Lecture Notes in Computer Science*. Springer-Verlag, July 1997.

[Var98] M.Y. Vardi. Reasoning about the past with two-way automata. In K. Larsen, S. Skyum, and G. Winskel, editors, *Proc. 25th Intl. Colloq. Aut. Lang. Prog.*, Lecture Notes in Computer Science, pages 628–641, Aalborg, Denmark, 1998. Springer-Verlag.

Necessary and Sufficient Assumptions for Non-interactive Zero-Knowledge Proofs of Knowledge for All NP Relations*

(Extended Abstract)

Alfredo De Santis[1], Giovanni Di Crescenzo[2], and Giuseppe Persiano[1]

[1] Dipartimento di Informatica ed Applicazioni,
Università di Salerno, 84081 Baronissi (SA), Italy
{ads,giuper}@dia.unisa.it

[2] Telcordia Technologies Inc., Morristown, NJ, 07960, USA
(Part of this work done while visiting Università di Salerno.)
giovanni@research.telcordia.com

* Copyright © 2000. Telcordia Technologies, Inc. All Rights Reserved.

Abstract. Establishing relationships between primitives is an important area in the foundations of Cryptography. In this paper we consider the primitive of non-interactive zero-knowledge proofs of knowledge, namely, methods for writing a proof that on input x the prover knows y such that relation $R(x, y)$ holds. These proofs have important applications for the construction of cryptographic protocols, as cryptosystems and signatures that are secure under strong types of attacks. They were first defined in [10], where a sufficient condition for the existence of such proofs for all NP relations was given. In this paper we show, perhaps unexpectedly, that such condition, based on a variant of public-key cryptosystems, is also necessary. Moreover, we present an alternative and natural condition, based on a variant of commitment schemes, which we show to be necessary and sufficient as well for the construction of such proofs. Such equivalence also allows us to improve known results on the construction of such proofs under the hardness of specific computational problems. Specifically, we show that assuming the hardness of factoring Blum integers is sufficient for such constructions.

1 Introduction

Understanding the conditions under which cryptographic applications are possible is of crucial importance for both the theoretical development of Cryptography and the concrete realization of real-life cryptographic systems. Modern complexity-based Cryptography is based on intractability assumptions such as the existence of several basic primitives, the most basic being perhaps that of the existence of one-way functions (functions that can be evaluated in polynomial time but that no polynomial time algorithm can invert). Since the first days of

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 451–462, 2000.
© Springer-Verlag Berlin Heidelberg 2000

modern Cryptography a main research area has been to establish relationships between all primitives used in this field. In some cases, researchers have been successful in presenting necessary and sufficient conditions for the existence of such primitives. For instance, it is known that the existence of one-way functions is necessary ([20]) and sufficient for the construction of other primitives as pseudo-random generators [19], commitment schemes [19,23], pseudo-random functions [19,14], signatures schemes [27,24], zero-knowledge proofs of membership for all languages in NP and zero-knowledge proofs of knowledge for all NP relations [16,19,23,26]. Moreover, some other primitives, such as public-key cryptosystems [9], are known to exist under the necessary and sufficient assumption of the existence of one-way functions with trapdoors.

The object of this paper is to consider the primitive of non-interactive zero-knowledge (nizk) proofs of knowledge. These proofs, first defined in [10], have important applications for the construction of cryptographic protocols with strong security, as public-key cryptosystems secure under chosen ciphertext attack [25] and digital signatures secure under existential forgery [4]. In [10] a sufficient condition was presented for the existence of nizk proofs of knowledge for all NP relations, and it was given strong evidence that it is not possible to construct nizk proofs of knowledge using one-way functions only.

Summary of main contributions. We study necessary and sufficient conditions for the existence of nizk proofs of knowledge for all NP relations. Our main contributions can be summarized as follows:

In [10] it is shown that the existence of *dense secure cryptosystems* and of *non-interactive zero-knowledge proofs of membership for all languages in NP* is sufficient for the existence of nizk proofs of knowledge for all NP relations. Here, we show that, perhaps unexpectedly, such assumption is also necessary.

We introduce the concept of an *extractable commitment* (a natural concept, dual to those of equivocable commitment, used in [3,11,12] and chameleon commitments of [8]) and show that the notions of extractable commitments and dense-secure cryptosystems are equivalent. It follows that the existence of extractable commitments and of nizk proofs of membership for all languages in NP is necessary and sufficient for the existence of nizk proofs of knowledge for all NP relations.

We then show that assuming difficulty of factoring Blum integers is sufficient for constructing extractable commitments and thus nizk proofs of knowledge. Previous constructions of nizk proofs of knowledge were based on provably non-weaker assumptions of the difficulty of inverting the RSA function and of deciding the quadratic residuosity predicate modulo Blum integers.

Organization of the paper. Definitions are in Section 2, the sufficient condition based on extractable commitment schemes is given in Section 3, the sufficient condition based on the hardness of factoring Blum integers is in Section 4, the proof that the two conditions based on dense cryptosystems and extractable commitment schemes are also necessary for the construction of nizk proofs of knowledge for all NP relations is in Section 5.

Basic notations, definitions and most proofs are omitted for lack of space.

2 Definitions

We present the definitions of cryptographic primitives of interest in this paper. We review nizk proofs of membership, nizk proofs of knowledge and dense encryption schemes and define the new notion of extractable commitment scheme.

Nizk proofs of membership. A nizk proof system of membership for a certain language L is a pair of algorithms, a prover and a verifier, the latter running in polynomial time, such that the prover, on input string x and a public random string, can compute a proof that convinces the verifier that the statement '$x \in L$' is true, without revealing anything else. Such proof systems satisfy three requirements: completeness, soundness and zero-knowledge. Completeness states that if the input x is in the language L, with high probability, the string computed by the prover makes the verifier accept. Soundness states that the probability that any prover, given the reference string, can compute an x not in L and a string that makes the verifier accept, is very small. Zero-knowledge is formalized by saying that there exists an efficient algorithm that generates a pair which has distribution indistinguishable from the reference string and the proof in a real execution of the proof system (see [5] for a formal definition).

Nizk proofs of knowledge. A nizk proof system of knowledge for a certain relation R is a pair of algorithms, a prover and a verifier, the latter running in polynomial time, such that the prover, on input string x and a public random string, can compute a proof that convinces the verifier that 'he knows y such that $R(x, y)$'. It is defined as a nizk proof system of membership for language $dom\, R$ that satisfies the following additional requirement, called *validity*: there exists an efficient extractor that is able to prepare a reference string (together with some additional information) and extract a witness for the common input from any accepting proof given by the prover; moreover, this happens with essentially the same probability that such prover makes the verifier accept. The formal definition from [10] follows.

Definition 1. Let P a probabilistic Turing machine and V a deterministic polynomial time Turing machine. Also, let R be a relation and L_R be the language associated to R. We say that (P,V) is a *non-interactive zero-knowledge proof system of knowledge for* R if it is a nizk proof of membership for L_R and if there exists a pair (E_0, E_1) of probabilistic polynomial time algorithms such that for all algorithms P', for all constants c and all sufficiently large n, $p_0 - p_1 \leq n^{-c}$, where p_0, p_1 are, respectively,

$$\mathrm{Prob}[\,(\sigma, aux) \leftarrow E_0(1^n); (x, Proof) \leftarrow P'(\sigma); w \leftarrow E_1(\sigma, aux, x, Proof) \;:\; (x, w) \in R\,];$$
$$\mathrm{Prob}[\,\sigma \leftarrow \{0,1\}^*; (x, Proof) \leftarrow P'(\sigma) : V(\sigma, x, Proof) = 1\,].$$

Public-key cryptosystems. We now review the definition of public-key cryptosystems, as first defined in [17]. A *public-key cryptosystem* is a triple of polynomial time algorithms (KG,E,D). The algorithm KG, called the *key-generation algorithm*, is probabilistic; on input a security parameter 1^n and a random string

r of appropriate length, KG returns a pair (pk, sk), where pk is called the *public-key*, and sk is called the *secret-key*. The algorithm E, called the *encryption algorithm*, is probabilistic; on input pk, a bit b, called the *plaintext*, and a random string s of appropriate length, E returns a string ct, called the *ciphertext*. The algorithm D, called the *decryption algorithm*, is deterministic; on input sk, pk and a string ct, D returns a bit b, or a special failure symbol $\perp$. The algorithms KG,E,D have to satisfy the two requirements of correctness and security, defined as follows. *Correctness* means that for all constants c, all sufficiently large n, and all $b \in \{0,1\}$, $p_b \geq 1 - n^{-c}$, where p_b is equal to

$$\text{Prob}\left[\, r, s \leftarrow \{0,1\}^*; (pk, sk) \leftarrow \text{KG}(1^n, r); ct \leftarrow \text{E}(pk, s, b); b' \leftarrow \text{D}(pk, sk, ct) : b' = b \,\right].$$

Security means that for any probabilistic polynomial time algorithm A', for all constant c, and any sufficiently large n, $q_0 - q_1$ is negligible, where

$$q_b = \text{Prob}\left[\, r, s \leftarrow \{0,1\}^*; (pk, sk) \leftarrow \text{KG}(1^n, r); ct \leftarrow \text{E}(pk, s, b) \;:\; A'(pk, ct) = b \,\right].$$

Dense public-key cryptosystems. A dense public-key cryptosystem is defined starting from a public-key cryptosystem, and performing the following two variations. First, the public key output by the key-generation algorithm is uniformly distributed over a set of the type $\{0,1\}^k$, for some integer k. Second, the security requirement holds for at least a noticeable[1] set of public keys output by the key-generation algorithm, rather than for all of them.

Definition 2. Let (KG,E,D) be a triple containing the above defined algorithms, as follows; namely, KG is a key-generation algorithm, E is an encryption algorithm and D is a decryption algorithm. Also, let (KG,E,D) satisfy the above defined correctness requirement. We say that (KG,E,D) is a *δ-dense public-key cryptosystem* if the following two requirements hold:

1. *Uniformity.* For all $n \in \mathcal{N}$, and all constants c, it holds that

$$\sum_{\alpha} |\, \text{Prob}[\, \alpha \leftarrow D_n \,] - \text{Prob}[\, \alpha \leftarrow U_n \,] \,| \leq n^{-c},$$

   where $D_n = \{r \leftarrow \{0,1\}^*; (pk, sk) \leftarrow \text{KG}(1^n, r) \;:\; pk\}$ and $U_n = \{r \leftarrow \{0,1\}^*; (pk, sk) \leftarrow \text{KG}(1^n, r); s \leftarrow \{0,1\}^{|pk|} \;:\; s\}$.
2. *δ-Security.* For any $m \in \mathcal{N}$, there exists a set $Hard_m \subseteq \{0,1\}^{q(m)}$ of size $\delta(m) \cdot 2^{q(m)}$, for some noticeable function δ, such that for any probabilistic polynomial time algorithm $A = (A_1, A_2)$, for any constant c, any sufficiently large n, and any $r \in Hard_n$,

$$\begin{aligned} \text{Prob}\,[\; & (pk, sk) \leftarrow \text{KG}(1^n, r); (m_0, m_1, aux) \leftarrow A_1(pk); b \leftarrow \{0,1\}; \\ & c \leftarrow \text{E}(pk, m_b); d \leftarrow A_2(pk, c, aux) : d = b \;\;] \;\; < 1/2 + n^{-c}. \end{aligned}$$

Commitment schemes. A *bit commitment scheme* (A,B) in the public random string model is a two-phase interactive protocol between two probabilistic polynomial time parties A and B, called the committer and the receiver, respectively,

[1] A function $f : \mathcal{N} \to \mathcal{N}$ is noticeable if there exists a constant c such that for all sufficiently large n, $f(n) \geq n^{-c}$.

such that the following is true. In the first phase (the commitment phase), given the public random string σ, A commits to a bit b by computing a pair of keys (com, dec) and sending com (the commitment key) to B. In the second phase (the decommitment phase) A reveals the bit b and the key dec (the decommitment key) to B. Now B checks whether the decommitment key is valid; if not, B outputs a special string $\perp$, meaning that he rejects the decommitment from A; otherwise, B can efficiently compute the string s revealed by A. (A,B) has to satisfy three requirements: correctness, security and binding.The *correctness* requirement states that the probability that algorithm B, on input σ, com, dec, can compute the committed bit b after the reference string σ has been uniformly chosen, and after algorithm A, on input σ, b, has returned pair (com, dec), is at least $1 - \epsilon$, for some negligible function ϵ. The *security* requirement states that given just σ and the commitment key com, no polynomial-time receiver B$'$ can distinguish with probability more than negligible whether the pair (σ, com) is a commitment to 0 or to 1. The *binding* requirement states that after σ has been uniformly chosen, for any algorithm A$'$ returning three strings com', dec_0, dec_1, the probability that dec_b is a valid decommitment for σ, com' as bit b, for $b = 0, 1$, is negligible.

Extractable Commitment Schemes. A commitment scheme is *extractable* if there exists an efficient extractor algorithm that is able to prepare a reference string (together with some additional information) and extract the value of the committed bit from any valid commitment key sent by the prover; here, by a valid commitment key we mean a commitment key for which there exists a decommitment key that would allow the receiver to obtain the committed bit. A formal definition follows.

Definition 3. Let (A,B) be a commitment scheme. We say that (A,B) is *extractable* if there exists a pair (E_0, E_1) of probabilistic polynomial time algorithms such that for all algorithms A', for all $b \in \{0, 1\}$, for all constants c and all sufficiently large n, it holds that $p_0 - p_1 \leq n^{-c}$, where p_0, p_1 are, resp.,

$$\text{Prob}[\,(\sigma, aux) \leftarrow E_0(1^n); (com, dec) \leftarrow A'(\sigma); b \leftarrow E_1(\sigma, aux, com) : B(\sigma, com, dec) = b\,];$$
$$\text{Prob}\left[\,\sigma \leftarrow \{0,1\}^*; (com, dec) \leftarrow A'(\sigma, b) : B(\sigma, com, dec) = b\,\right].$$

3 Sufficient Conditions for Nick Proofs of Knowledge

In this section we present an additional sufficient condition for the construction of nizk proofs of knowledge for any NP relation. Formally, we obtain the following

Theorem 1. Let R be a polynomial time computable relation and let L_R be the language associated to R. If there exists a nizk proof system of membership for L_R and there exists an extractable commitment scheme then it is possible to construct a nizk proof of knowledge for R.

In order to prove the above theorem, we start by showing that the following two cryptographic primitives are equivalent: extractable commitment schemes and dense public-key cryptosystems. More precisely, we show the following

Lemma 1. Let δ be a noticeable function. There exists (constructively) an extractable commitment scheme if and only if there exists (constructively) a δ-dense public-key cryptosystem.

Theorem 1 follows then directly from Lemma 1 and the main result in [10], saying that the existence of dense public-key cryptosystems and the existence of nizk proofs of membership for all languages in NP implies the existence of a nizk proof of knowledge for all polynomial time relations R. The proof of Lemma 1 is described in Sections 3.1 and 3.2. We note that one could define a notion of δ-dense trapdoor permutations (by modifying the definition of trapdoor permutations as one modifies the definition of public-key cryptosystems to define their δ-dense variant). As a direct corollary of the techniques in this paper, and using the fact that the existence of trapdoor permutations is sufficient for the existence of nizk proofs of membership for all languages in NP [13], we would obtain that the existence of δ-dense trapdoor permutation (alone) is a sufficient condition for the existence of nizk proofs of knowledge for all NP relations. We now describe two corollaries about constructing extractable commitment schemes using both general complexity-theoretic conditions and also practical conditions based on the conjectured hardness of specific computational problems. In the first case we have a negative result and in the second some positive ones.

Constructions under general conditions. We first observe that δ-dense public-key cryptosystems imply the existence of $(1-\epsilon)$-dense public-key cryptosystems, where ϵ is a negligible function (this follows by using independent repetition of the δ-dense cryptosystem, as in Yao's XOR theorem [28]). Then, using the result in [21] saying that constructing $(1-\epsilon)$-dense public-key cryptosystems based on one-way permutations only is as hard as proving that P $\neq$ NP, and our Lemma 1, we obtain the following

Corollary 1. Constructing extractable commitment schemes based on one-way permutations only is as hard as proving that P $\neq$ NP.

We note that the above result is especially interesting in light of the fact that, based on one-way functions only, it is known how to construct commitment schemes (using results from [19,23]). In other words, this shows that constructing extractable commitment schemes from any commitment scheme can be as hard as proving that P $\neq$ NP.

Constructions under practical conditions. Combining Lemma 1 with results in [10], we obtain the following

Corollary 2. Let δ be a noticeable function. If breaking the RSA cryptosystem, or the decisional Diffie Hellman problem or deciding quadratic residuosity is hard, then it is possible to construct an extractable commitment scheme.

3.1 From Extractable Commitments to Dense Cryptosystems

In this subsection we prove one direction of the main statement in Lemma 1. Given an extractable commitment scheme (A,B,(E_0,E_1)), we construct a δ-dense public-key cryptosystem (KG,E,D), for some noticeable function δ.

Constructing the dense cryptosystem. The basic idea of the construction of the dense public-key cryptosystem is to use the extractor both for the key-generation and the decryption algorithms. Specifically, the key-generation algorithm KG is obtained by running the first extractor algorithm E_0, and outputs the reference string σ returned by E_0 as a public key and the private information aux as a secret key. The encryption of message m is obtained by performing a commitment to m using algorithm A and the public key output by KG as a reference string. The decryption is obtained by running the second extractor algorithm E_1 on input the ciphertext returned by the encryption algorithm, and the public and secret keys returned by KG. A formal description follows.

The algorithm KG. On input a security parameter 1^n and a sufficiently long random string r, algorithm KG sets $(\sigma, aux) = E_0(1^n, r)$, $pk = \sigma$, $sk = aux$ and outputs: (pk, sk).

The algorithm E. On input a public key pk and a message m, algorithm E sets $\sigma = pk$, $(com, dec) = A(\sigma, m)$, $ct = com$, and outputs: ct.

The algorithm D. On input public key pk, secret key sk and ciphertext ct, algorithm D sets $\sigma = pk, aux = sk, com = ct, mes = E_1(\sigma, aux, com)$ and outputs: mes.

3.2 From Dense Cryptosystems to Extractable Commitments

In this subsection we prove the other direction of Lemma 1. Given a δ-dense public-key cryptosystem (KG,E,D), for some noticeable function δ, we construct an extractable commitment scheme (A,B,(E_0,E_1)).

Constructing the extractable commitment scheme. The basic idea of the construction of the extractable commitment scheme is to use a portion of the reference string as specifying several public keys for the encryption scheme, and to commit to a bit b by encrypting such bit according to any of such public keys. The first extractor algorithm would prepare the reference string by running the key generator algorithm several times, by keeping the secret keys private and setting the reference string equal to the sequence of public keys thus obtained. The second extractor algorithm would run the decryption algorithm to decrypt all encryptions that are in the commitment key; if all such decryptions return the same bit, then the algorithm outputs such bit. Let $m = n/\delta(n)$.

The algorithm A. On input reference string σ and bit b, algorithm A does the following. First, A writes σ as $\sigma = \sigma_1 \circ \cdots \circ \sigma_m$, where $|\sigma_i| = n$, for $i = 1, \ldots, m$. Then, A sets $pk_i = \sigma_i$ and computes $ct_i = E(pk_i, b)$ using a uniformly chosen string r_i as a random string for algorithm E, for $i = 1, \ldots, m$. Finally E sets $com = (ct_1, \ldots, ct_m)$ and $dec = (r_1, \ldots, r_m)$, and outputs: (com, dec).

The algorithm B. On input reference string σ, and strings com, dec, algorithm B does the following. First B writes σ as $\sigma = pk_1 \circ \cdots \circ pk_m$, com as $com = (com_1, \ldots, com_m)$, and $dec = (dec_1, \ldots, dec_m)$, Then B verifies that there exists a bit b such that $com_i = E(pk_i, b)$ using dec_i as a random string. If all these verifications are successful then B outputs: b, else B outputs: $\perp$.

The algorithm $\mathbf{E}_0$. On input 1^n, algorithm E_0 does the following. E_0 lets $(pk_i, sk_i) = \mathrm{KG}(1^n)$, for $i = 1, \ldots, m$, and using each time independently chosen random bits. Finally E_0 sets $\sigma = pk_1 \circ \cdots \circ pk_m$ and $aux = (sk_1, \ldots, sk_m)$, and outputs: (σ, aux).

The algorithm $\mathbf{E}_1$. On input reference string σ, and strings com, aux, algorithm E_1 does the following. First algorithm E_1 writes σ as $\sigma = pk_1 \circ \cdots \circ pk_m$, com as $com = (com_1, \ldots, com_m)$, and aux as $aux = (sk_1, \ldots, sk_m)$. Then E_1 checks if there exists a bit b such that $b = \mathrm{D}(pk_i, sk_i, com_i)$, for $i = 1, \ldots, m$. If so, then E_1 outputs: b; if not, then E_1 outputs: $\perp$.

4 A Practical Sufficient Condition: Hardness of Factoring

In this section we consider the problem of constructing nizk proofs of knowledge for all NP relations based on practical conditions, namely, using the conjectured hardness of specific computational problems. We show that the hardness of factoring Blum integers (BF assumption) is sufficient for this task.

Theorem 2. If the BF assumption holds then it is possible to construct a nizk proof system of knowledge for all polynomial-time relations.

The theorem follows from a construction of δ-secure extractable commitment schemes from the BF assumption, shown in Section 4.1, an application of Lemma 1, and the result in [13], which implies that the BF assumption is sufficient for constructing a nizk proof of membership for all languages in NP. As another application of this fact and Lemma 1, we obtain:

Corollary 3. If the BF assumption holds then it is possible to construct a δ-dense public-key cryptosystem, for some noticeable function δ.

4.1 An Extractable Commitment Based on Factoring

We now present a construction of an extractable commitment scheme based on the hardness of factoring Blum integers (we remark that none of the previous commitment schemes based on such assumptions is extractable).

Constructing the extractable commitment scheme. We informally describe the basic ideas of our construction. First, a portion of the reference string is used as an integer x that is a Blum integer (note that by the results on the distribution of primes, it follows that the probability that a uniformly chosen n-bit integer satisfies this property is at least $\Omega(1/n^2)$ and therefore a noticeable function). Then the squaring function is used in order to hide a value r

uniformly chosen among the elements in Z_x^* smaller than $x/2$ (note that this mapping defines a one-to-one function). Then, if b is the bit to be committed, the committer chooses an n-bit random string s such that $r \odot s = c$ and determines $(s, r^2 \bmod x, c \oplus b)$ as a commitment key and r as a decommitment key (here, the $\odot$ operation is used to define a hard-core bit, as from the result of [15]). Two levels of repetitions are required: the first ensures that the probability that two square roots of the same integer satisfy the $\odot$ equation for different values of c; the second ensures that at least one modulus from the public random string is a Blum integer. Now we formally describe the scheme. We will denote by S the probabilistic polynomial time algorithm from [1] which, on input 1^n, outputs an n-bit integer x, together with its factorization $fact$.

The algorithm A: On input the reference string σ and a bit b, algorithm A does the following. A writes σ as $\sigma = \sigma_1 \circ \cdots \circ \sigma_m$, for $m = n^3$, and repeats the following for each $j = 1, \ldots, m$ (using independently chosen random bits). A writes σ_j as $\sigma_j = x_j \circ s_{1j} \circ \cdots \circ s_{2n,j}$, where $x_j, s_{1j}, \ldots, s_{2n,j} \in \{0,1\}^n$. Then A randomly chooses $r_{1j}, \ldots, r_{2n,j} \in Z_{x_j}^*$ such that $r_{1j}, \ldots, r_{2n,j} \leq x_j/2$. Now, for $i = 1, \ldots, 2n$, A computes $c_{ij} = s_{ij} \odot r_{ij}$, if $b = 0$, and $c_{ij} = 1 - (s_{ij} \odot r_{ij})$, if $b = 1$. Finally A computes $z_{ij} = r_{ij}^2 \bmod x$, for $i = 1, \ldots, 2n$, sets $com_j = ((z_{1j}, s_{1j}, c_{1j}), \ldots, (z_{2n,j}, s_{2n,j}, c_{2n,j}))$ and $dec_j = (r_{1j}, \ldots, r_{2n,j})$, $com = (com_1, \ldots, com_{n^3})$ and $dec = (dec_1, \ldots, dec_{n^3})$ and outputs: (com, dec).

The algorithm B: On input the reference string σ, and two strings com, dec, algorithm B does the following. B writes σ as $\sigma = \sigma_1 \circ \cdots \circ \sigma_m$, for $m = n^3$, and repeats the following for each $j = 1, \ldots, m$ (using independently chosen random bits). B writes each σ_j as $\sigma_j = x_j \circ s_{1j} \circ \cdots \circ s_{2n,j}$, where $x_j, s_{1j}, \ldots, s_{2n,j} \in \{0,1\}^n$. Then B checks that com can be written as $com = (com_1, \ldots, com_{n^3})$, and dec can be written as $dec = (dec_1, \ldots, dec_{n^3})$, where $com_j = ((z_{1j}, s_{1j}, c_{1j}), \ldots, (z_{2n,j}, s_{2n,j}, c_{2n,j}))$, $dec_j = (r_{1j}, \ldots, r_{2n,j})$, $z_{ij}, r_{ij} \in Z_{x_j}^*$, $c_{ij} \in \{0,1\}$, $r_{ij} \leq x_j/2$, and $z_{ij} = r_{ij}^2 \bmod x_j$, for $j = 1, \ldots, n^3$ and $i = 1, \ldots, 2n$. If any of the above verifications is not satisfied, then B outputs: $\perp$ and halt. Otherwise, for each $j = 1, \ldots, n^3$, B checks that either $s_{ij} \odot r_{ij} = c_{ij}$, for $i = 1, \ldots, 2n$, in which case B sets $b_i = 0$ or $s_{ij} \odot r_{ij} = 1 - c_{ij}$, for $i = 1, \ldots, 2n$, in which case B sets $b_i = 1$. If $b_1 = \cdots = b_{2n}$ then B sets $b = b_1$ else B sets $b = \perp$. Finally, B outputs: b.

The algorithm $\mathbf{E}_0$: On input 1^n, algorithm E_0 does the following. E_0 runs n^3 times algorithm S on input 1^n (each time using independently chosen random bits), and lets $((x_1, fact_1), \ldots, (x_{n^3}, fact_{n^3}))$ be its output. Then E_0 sets $\sigma = x_1 \circ \cdots \circ x_{n^3}$, sets $aux = (fact_1, \ldots, fact_{n^3})$, and outputs: (σ, aux).

The algorithm $\mathbf{E}_1$: On input σ, com, aux, algorithm E_1 does the following. E_1 writes σ as $\sigma_1, \ldots, \sigma_{n^3}$, and each σ_j as $\sigma_j = x_j \circ s_{1j} \circ \cdots \circ s_{2n,j}$, where x_j is an n-bit integer, and com as $com = com_1 \circ \cdots \circ com_{n^3}$, where $com_j = ((z_{1j}, s_{1j}, c_{1j}), \ldots, (z_{2n,j}, s_{2n,j}, c_{2n,j}))$, for $z_{ij} \in Z_x^*$, $s_{ij} \in \{0,1\}^n$ and $c_{ij} \in \{0,1\}$, for $i = 1, \ldots, 2n$ and $j = 1, \ldots, n^3$. E_1 checks that com can be written as above and that aux_j is the factorization of x_j, for $j = 1, \ldots, n^3$. If these verifications are not satisfied, E_1 outputs: $\perp$ and halts. Otherwise, E_1 uses aux to compute $r_{1j}, \ldots, r_{2n,j} \in Z_x^*$ such that $r_{ij} \leq x/2$, $r_{ij}^2 = z_{ij} \bmod x$, and it holds that

$r_{ij} \odot s_{ij} = c_{ij} \oplus b'$, for $i = 1, \ldots, 2n$, $j = 1, \ldots, n^3$ and $b' \in \{0, 1\}$. E_1 sets $b = b'$ if such a b' exists, or $b = \perp$ otherwise, and outputs: b.

5 Necessary and Sufficient Conditions

In this section we consider the problem of providing necessary conditions for the construction of nizk proofs of knowledge for any NP relation. We show that both sufficient conditions considered in Section 3 are also necessary, and therefore equivalent. Formally, we obtain the following

Theorem 3. Let δ be a noticeable function, let R be a polynomial time computable relation, let L_R be the language associated to R, and assume that $L_R \notin$ BPP. The following three conditions are equivalent:

1. the existence of a nizk proof of knowledge for R;
2. the existence of a δ-dense public-key cryptosystem and the existence of a nizk proof system of membership for L_R;
3. the existence of an extractable commitment scheme and the existence of a nizk proof system of membership for L_R.

First of all we remark that the assumption made in the above theorem that $L_R \notin$ BPP is wlog. Indeed, if L_R is in BPP, then a trivial and unconditional construction for a nizk proof of knowledge for R exists (namely, the prover sends no proof to the verifier who can compute by himself a y such that $R(x, y)$ holds, for any input x). Therefore, we continue our discussion by assuming that $L_R \notin$ BPP. In order to prove the above theorem, it is enough to show that the one of the two conditions shown to be sufficient in Section 3 is indeed necessary as well. We do this in the rest of the section with extractable commitment schemes. We divide the presentation in two steps. Since the existence of nizk proofs of knowledge for a relation R implies, by definition, the existence of a nizk proof of membership for the associated language L_R, it is enough to prove that 1) the existence of nizk proofs of knowledge for a relation R associated to a language not in BPP implies the existence of commitment schemes, and 2) the existence of commitment schemes and of nizk proofs of knowledge for all polynomial time relations implies the existence of extractable commitment schemes. Clearly, Theorem 3 follows by combining these statements.

Nizk proofs of knowledge imply commitment schemes. We first discuss the possibility of proving that the existence of a nizk proof of knowledge for relation R implies the existence of a one-way function. Notice that we cannot use the result of [26] since it would give such a result by making the additional assumption that the language L_R is hard on average; namely, it is not in average-BPP (which is likely to be a class larger than BPP). Instead, we give the following direct construction. Let (P,V) be the nizk proof of knowledge for R and let (E_0,E_1) be the extractor associated to (P,V). Moreover, let $m(n)$ be an upper bound on the number of random bits that E_0 uses when taking as input security parameter 1^n. Since $m(n)$ can be chosen as a polynomial, there exists a function $l : \mathcal{N} \to \mathcal{N}$ such that $m(l(n)) = n$ for all $n \in \mathcal{N}$, and l is itself bounded by a

polynomial. Then, we define the function ensemble $f = \{f_n\}_{n\in\mathcal{N}}$, where for all $x \in \{0,1\}^n$, it holds that $f_n(x) = \sigma$, such that $(\sigma, aux) = E_0(1^{l(n)}, x)$. We can prove that if L_R is not in BPP then f is a one-way function (intuitively, observe that if this is not the case, then V can invert f and use the proof received by P in order to obtain w with some not negligible probability, thus contradicting the zero-knowledge property of (P,V)). A construction of a commitment scheme in the public random string model starting from any one-way function can be obtained by using the results in [19,23] and observing that the commitment scheme in [23] can be modified so to require only one move in the public random string setting.

Commitment schemes and nizk proofs of knowledge imply extractable commitment schemes. We assume the existence of a commitment scheme (C,D) in the public random string model and the existence of a nizk proof of knowledge (P,V) for any polynomial time relation R, with an associated extractor (Ext_0,Ext_1). We would like to construct an extractable commitment scheme (A,B,(E_0,E_1)). The basic idea of our construction is to have algorithm A commit to a bit b by using algorithm C and then using algorithm P to prove the knowledge of a valid decommitment key for the commitment key output by C.

The algorithm A: On input the reference string σ and a bit b, algorithm A does the following. A writes σ as $\sigma = \sigma_1 \circ \sigma_2$, sets $(com_1, dec_1) = C(\sigma_1, b)$ and defines relation $R_2 = R_2(\sigma, b) = \{(c,d) \,|\, D(\sigma, c, d) = b\}$. Then A runs algorithm P using com_1 as public input, dec_1 as private input, and σ_2 as a reference string in order to compute a proof of knowledge $proof$ of dec_1 such that (com_1, dec_1) belong to relation R_2. Finally A sets $com = (com_1, proof)$ and $dec = dec_1$ and outputs: (com, dec).

The algorithm B: On input the reference string σ, and two strings com, dec, algorithm B does the following. B writes σ as $\sigma = \sigma_1 \circ \sigma_2$ and com as $com = (com_1, proof)$, and defines relation $R_2 = R_2(\sigma, b) = \{(c,d) \,|\, D(\sigma, c, d) = b\}$. Then B verifies that $\mathrm{D}(\sigma_1, com_1, dec) \neq \perp$, and $V(\sigma_2, com_1, proof) = 1$. If any of the above verifications is not satisfied B outputs: $\perp$ else B computes $b = \mathrm{D}(\sigma_1, com_1, dec)$ and outputs: b.

The algorithm $\mathbf{E}_0$: On input 1^n, algorithm E_0 uniformly chooses σ_1, computes $(\sigma_2, aux) = \mathrm{Ext}_0(1^n)$ and sets $\sigma = \sigma_1 \circ \sigma_2$. Finally E_0 outputs: (σ, aux).

The algorithm $\mathbf{E}_1$: On input σ, com, aux, algorithm E_1 writes σ as $\sigma_1 \circ \sigma_2$ and com as $com = (com_1, proof)$. Then E_1 sets $dec_1 = \mathrm{Ext}_1(\sigma_2, com_1, proof, aux)$ and computes $b = \mathrm{D}(\sigma_1, com_1, dec_1)$. Finally E_1 outputs: b.

References

1. E. Bach, *How to Generated Random Factored numbers*, SIAM Journal on Computing, vol. 17, n. 2, 1988.
2. E. Bach and J. Shallit, *Algorithmic Number Theory*, MIT Press, 1996.
3. D. Beaver, *Adaptive Zero-Knowledge and Computational Equivocation*, in Proc. of FOCS 96.

4. M. Bellare and S. Goldwasser, *Methodology for Constructing Signature Schemes based on Non-Interactive Zero-Knowledge Proofs*, in Proc. of CRYPTO 88.
5. M. Blum, A. De Santis, S. Micali, and G. Persiano, *Non-Interactive Zero-Knowledge*, SIAM Journal of Computing, vol. 20, no. 6, Dec 1991, pp. 1084–1118.
6. M. Blum, P. Feldman, and S. Micali, *Non-Interactive Zero-Knowledge and Applications*, Proc. of STOC 88.
7. M. Blum and S. Micali, *How to Generate Cryptographically Strong Sequence of Pseudo-Random Bits*, SIAM J. on Computing, vol. 13, no. 4, 1984, pp. 850–864.
8. G. Brassard, C. Crépeau, and D. Chaum, *Minimum Disclosure Proofs of Knowledge*, Journal of Computer and System Sciences, vol. 37, no. 2, pp. 156–189.
9. W. Diffie and M. Hellman, *New Directions in Cryptography*, in IEEE Transaction in Information Theory, 22, 1976.
10. A. De Santis and P. Persiano, *Zero-Knowledge Proofs of Knowledge without Interaction*, in Proc. of FOCS 92.
11. G. Di Crescenzo, Y. Ishai, and R. Ostrovsky, *Non-Interactive and Non-Malleable Commitment,* in Proc. of STOC 98.
12. G. Di Crescenzo and R. Ostrovsky, *On Concurrent Zero-Knowledge with Pre-Processing,* in Proc. of CRYPTO 99.
13. U. Feige, D. Lapidot, and A. Shamir, *Multiple Non-Interactive Zero-Knowledge Proofs Based on a Single Random String*, in Proc. of STOC 90.
14. O. Goldreich, S. Goldwasser, and S. Micali, *How to Construct Random Functions*, Journal of the ACM, vol. 33, no. 4, 1986, pp. 792–807.
15. O. Goldreich and L. Levin, *A Hard-Core Predicate for any One-Way Function*, in Proc. of FOCS 90.
16. O. Goldreich, S. Micali, and A. Wigderson, *Proofs that Yield Nothing but their Validity or All Languages in NP Have Zero-Knowledge Proof Systems*, Journal of the ACM, vol. 38, n. 1, 1991, pp. 691–729.
17. S. Goldwasser, and S. Micali, *Probabilistic Encryption*, Journal of Computer and System Sciences, vol. 28, n. 2, 1984, pp. 270–299.
18. S. Goldwasser, S. Micali, and C. Rackoff, *The Knowledge Complexity of Interactive Proof-Systems*, SIAM Journal on Computing, vol. 18, n. 1, 1989.
19. J. Hastad, R. Impagliazzo, L. Levin, and M. Luby, *Construction of a Pseudo-Random Generator from any One-Way Function*, SIAM Journal on Computing, vol. 28, n. 4, pp. 1364–1396, 1999.
20. R. Impagliazzo and M. Luby, *One-Way Functions are Necessary for Complexity-Based Cryptography*, in Proc. of FOCS 89.
21. R. Impagliazzo and S. Rudich, *Limits on the Provable Consequences of One-Way Permutations*, in Proc. of STOC 91.
22. M. Luby and C. Rackoff, *How to Construct a Pseudo-Random Permutation from a Pseudo-Random Function*, in SIAM Journal on Computing, vol. 17, n.2, Aug 1988.
23. M. Naor, *Bit Commitment using Pseudorandomness*, in Proc. of CRYPTO 91.
24. M. Naor and M. Yung, *Universal One-way Hash Functions and their Cryptographic Applications*, in Proc. of STOC 89.
25. M. Naor and M. Yung, *Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attack*, Proc. of STOC 90.
26. R. Ostrovsky and A. Wigderson, *One-way Functions are Necessary for Non-Trivial Zero-Knowledge Proofs*, in Proc. of ISTCS 93.
27. J. Rompel, *One-way Functions are Necessary and Sufficient for Secure Signatures*, in Proc. of STOC 90.
28. A. Yao, *Theory and Applications of Trapdoor Functions,* in Proc. of FOCS 82.

Fast Verification of Any Remote Procedure Call: Short Witness-Indistinguishable One-Round Proofs for NP

William Aiello[1], Sandeep Bhatt[2], Rafail Ostrovsky[3], and S. Raj. Rajagopalan[3]

[1] AT&T Labs-Research. aiello@research.att.com
[2] Akamai Technologies. bhatt@akamai.com
[3] Telcordia Technologies. {rafail,sraj}@research.telcordia.com

Abstract. Under a computational assumption, and assuming that both Prover and Verifier are computationally bounded, we show a one-round (i.e., Verifier speaks and then Prover answers) witness-indistinguishable interactive proof for NP with poly-logarithmic communication complexity. A major application of our main result is that we show how to check in an efficient manner and without any additional interaction the correctness of the output of *any* remote procedure call.

1 Introduction

Under a computational assumption, and assuming that both Prover and Verifier are computationally bounded, we show a one-round (i.e., Verifier speaks and then Prover answers) witness-indistinguishable interactive proof for NP with poly-logarithmic communication complexity. That is, the properties of our one-round interactive proof (argument) are as follows:

- *Perfect Completeness*: On a common input $x \in L$ for any L in NP if Prover is given a witness as a private input, it convinces Verifier (i.e. Verifier accepts) with probability one.
- *Computational Soundness*: If $x \notin L$, no poly-time cheating Prover can make the Verifier accept except with negligible probability.
- *Protocol is Short*: the communication complexity (of both messages) is poly-logarithmic in the length of the proof.
- *Witness-indistinguishability*: The interactive proof is *witness indistinguishable*: If there is more then one witness that $x \in L$ then for any two witnesses no poly-time cheating Verifier can distinguish which was given to a Prover as a private input.

Our result improves upon best previous results on short computationally-sound proofs (arguments) of Micali [20] and of Kilian [15,16] which required either three messages of interaction or the assumption of the existence of a random oracle. Our result also improves the communication-complexity of one-round witness-indistinguishable arguments ("Zaps") of Dwork and Naor [7] which required polynomial communication complexity in the length of the proof. One important difference in our protocol vs. [7] is that our protocol is private-coin protocol[1] where as the Dwork and Naor protocol is public-coin.

[1] Private-coin protocol is the one where Verifier tosses coins but keeps them *secret* from the prover, where is public-coin protocols are the ones where Verifier can just publish its coin-flips.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 463–474, 2000.
© Springer-Verlag Berlin Heidelberg 2000

A major corollary of our result is that we show how to check in an efficient manner and without any additional interaction the correctness of the output of *any* remote procedure call. That is, if Alice requests Bob to execute any remote code and give back the result, Alice can also provide to Bob (together with the code) a very short public-key and receive back from Bob not only the result of the execution but also a very short and easily[2] verifiable *certificate* that the output was computed correctly. We stress that our scheme for verifiable remote procedure call is round-optimal: there is only one message from Alice to Bob (i.e. the procedure call) and only one message back from Bob to Alice (with the output of the procedure call and the certificate of correctness). Our scheme guarantees that if Bob follows the protocol then Alice always accepts the output, but if the claimed output is incorrect Bob can not construct a certificate which Alice will accept, except with negligible probability. The public key and the certificate is only of poly-logarithmic size and can be constructed by Bob in polynomial time and verified by Alice in poly-logarithmic time. This resolves the major open problem posed Micali in the theory of short computationally-sound proofs [20] (i.e., that they exist with only one round of interaction under a complexity assumption). This also improves the works of [20,15,16,24] and also of Ergün, Kumar and Rubinfeld [12] who consider this problem for a restricted case of languages in NP with additional algebraic properties.

It is interesting to point out that the proof of our main result also shows that the heuristic approach of Biehl, Meyer and Wetzel [4] of combining Probabilistically Checkable Proofs of Arora at. al., [2] with Single-Database Private Information Retrieval of Kushilevitz and Ostrovsky [18] and of Cachin, Micali and Stadler [6] is valid, though with additional steps both in the protocol and in its proof of security. More specifically, our main result uses a novel combinatorial technique to prove computational soundness of our interactive proof and we believe this proof is of independent interest.

Our result also has interesting connection to the PCP theorem [2]. Recall that in the PCP settings Prover must send to the Verifier an entire proof. Subsequently verifier needs to read only a few bits of this proof at random in order to check its validity. Notice, however, that the communication complexity is large (since the Prover must send the entire PCP proof). Moreover, notice that even though verifier is interested in reading only a few bits, Verifier must *count* the received bits, in order to decide which bits to read. In our setting, verifier asks his question first and then Prover can make his answers dependent on the verifiers questions. Never-the-less, we show that even if computationally-bounded Prover gets to see the verifiers message before it generates its answer, it can not cheat the verifier, even if his answer depends on the question. Moreover, the length of the proof is dramatically shorter then the PCP proof.

Our main result have several implications including a different a method of reducing the error in one-round two-prover interactive proofs with small communication complexity. Our proof also has implications to the generalized PIR protocols, where user asks for many bits and wishes to know if there exist *some* database which is consistent with all the answers. We discuss this and other implications after we prove our main result.

[2] By very short and easily verifiable we mean that the size of the public-key/certificate as well as generation/verification time is a product of two quantities: a fixed polynomial in the security parameter times a term which is poly-logarithmic in the running time of the actual remote procedure call.

2 Preliminaries

First, we recall some of the terminology and tools that we use in our construction. For definitions and further discussion see provided references. We use standard notions of expected polynomial-time computations, polynomial indistinguishability, negligible fractions and hybrid arguments (for example, see Yao [25]) and the notion of witness-indistinguishability [11].

We shall use Single Database Private Information Retrieval (PIR) protocols first put forward by Kushilevitz and Ostrovsky [18] and further improved (to an essentially optimal communication complexity) by Cachin, Micali and Stadler [6]. (In the setting of multiple non-communicating copies of the database PIR was originally defined by Chor at. al. [5].) Recall that Single Database PIR are private-coin protocols between user and a database, where database has an n-bit string D and a user has an index $1 \leq i \leq n$. Assuming some computational (hardness) assumption [18,6,9], the user forms a PIR question to the database based on i and a security parameter[3] k, which database answers. The answer can be "decrypted" by the user to correctly learn the i'th bit of D. The key property is that i remains hidden from the database. That is, for any two indexes i and j the PIR questions of i and j are computationally indistinguishable (i.e., can not be distinguished except with negligible probability, denoted as ϵ_{PIR}.) By $PIR(cc, u, db)$ we denote any one-round single database PIR protocol with communication complexity $cc(n, k)$ the running time of the user (to both generate the question and decode the answer from the database) to be $u(n, k)$ and and running time of the database to be $db(n, k)$ (where $db(n, k)$ must be polynomial in n and k.)

We remark that by standard hybrid argument, poly-size vectors of PIR queries are also indistinguishable. We also point our that both [18] and [6] are one-round protocols, unlike [19]. Furthermore, we will also use Secure-PIR (SPIR) protocols in a single database setting (see Gertner at al. [13] and Kushilevitz and Ostrovsky [18]) and one-round SPIR protocol (based on any 1-round PIR protocol and one-round OT) of Naor and Pinkas [21]. The latter result shows how to convert any PIR protocol to a SPIR protocol using one call to PIR and an additive polylogarithmic overhead in communication.

We will use Probabilistically Checkable Proof (PCP) proofs of Arora at al. [2] and 3-query PCP proofs of Håstad [14] as well as holographic PCP proofs of Babai at al. [3] and Polishchuk and Spielman [22]. Recall that a language in NP has a (ϵ_{PCP}, k) PCP proof if two conditions hold: For every string x in the language there is a polynomially long "proof" (which, given a witness can be constructed in polynomial time) which the probabilistic polynomial time Verifier V will always accept. For every string not in the language there is no proof that such a V will accept with probability greater than ϵ_{pcp}. Furthermore, V reads only at most k bits. Additionally, we shall need Zero-Knowledge PCP of Kilian at al. [17].

3 Problem Statements and Our Results

We state our problem in two settings, the setting of proving an NP statement and the setting of efficient verification of remote procedure calls.

[3] By $k = k(\cdot)$ we denote a sufficiently large security parameter. That is, $k(\cdot)$ is a function of n so that for sufficiently large n, the error probability is negligible. In PIR implementations (i.e., [18,6]) depending on a particular assumption, $k(n)$ is typically assumed to be either $\log^{O(1)}(|n|)$ or $|n|^{\epsilon}$ for $\epsilon > 0$.

In the setting of proving an NP statement, both prover and verifier are probabilistic poly-time machines. They both receive as as a common input (i.e., on their work-tapes) an input x and a security parameter k. Here, since x is given on the work-tape of the verifier, we do not charge the length of x as part of communication complexity, nor do we charge the time to read x, as it is given before the protocol begins. Additionally Prover receives (if $x \in L$) as a private input a witness w that $x \in L$. W.l.o.g. we assume that $|w| \geq |x|$ (we stress here that the interesting case, and the one that we also handle, is the one where $|w| >> |x|$).

THEOREM 1 Assume that one-round $PIR(cc, db, u)$ scheme exist. Then, there exists (P,V) one-round computationally-sound proof with perfect completeness and computational soundness, so that: the communication complexity is $O(\log^{O(1)} |w| \cdot cc(|w|^{O(1)}, k))$; prover running time is $O(|w|^{O(1)} + db(|w|^{O(1)}, k))$; and verifier running time is $O(\log^{O(1)} |w| \cdot u(|w|^{O(1)}, k))$.

If we use Cachin at al. implementation of PIR based on ϕ-hiding assumption [6], and $k(x) = \log^{O(1)} |x|$ for sufficiently large x, then we can achieve *both* poly-logarithmic communication complexity and poly-logarithmic user's computation:

COROLLARY 1 Assume that ϕ-hiding assumption holds. Then, for any $\epsilon > 0$ there exists (P,V) one-round computationally-sound proof with perfect completeness and computational soundness, so that for x sufficiently large, the total communication complexity is $O(\log^{O(1)} |w|)$; prover computation is $|w|^{O(1)}$ and verifiers computation is $O(\log^{O(1)} |w|)$.

If we use Kushilevitz and Ostrovsky implementation of PIR based on quadratic residuosity assumption [18], and $k(x) = |x|^{\epsilon}$ for $\epsilon > 0$ and x sufficiently large, then we get the following:

COROLLARY 2 Assume that quadratic residuosity assumption holds. Then, there exists (P,V) one-round computationally-sound proof with perfect completeness and computational soundness, so that for x sufficiently large, the total communication complexity is $|w|^{\epsilon}$; prover computation is $|w|^{O(1)}$ and verifiers computation is $|w|^{\epsilon}$, for any $\epsilon > 0$.

We remark that using *holographic proofs* [3,22], we can make the running time of the prover in the above theorem and in both corollaries nearly-linear.

We also show how we can add witness-indistinguishability property to our low-communication complexity protocol by combining Zero-Knowledge PCP [17]) with one-round SPIR protocols [13,18,21]. That is, we modify our results above to have witness indistinguishability property:

THEOREM 2 Assume that one-round $PIR(cc, db, u)$ scheme exist. Then, there exists (P,V) one-round witness-indistinguishable computationally-sound proof with perfect completeness and computational soundness, so that: the communication complexity is $O(\log^{O(1)} |w| \cdot cc(|w|^{O(1)}, k))$; prover running time is $O(|w|^{O(1)} + db(|w|^{O(1)}, k))$; and verifier running time is $O(\log^{O(1)} |w| \cdot u(|w|^{O(1)}, k))$.

Our second setting is that of verification of any remote procedure call. Here, Alice has remote procedure call Π (an arbitrary code with an arbitrary input) and a polynomial bound t on the running time. Alice generates a public-key and

a private key, keeps the private key to herself and send the public-key, Π and t to Bob. Bob executes $y \leftarrow \Pi$ for at most t steps (y is defined as $\perp$ if the program Π did not terminate in t steps). Bob using public-key also computes a certificate c that y is correct and sends c and y back to Alice. Alice using her public-key and private-key decides to either accept or reject y. We say that the remote procedure call verification is *correct* if two conditions hold: if Bob follows the protocol then Alice always accepts; on the other hand no poly-time cheating Bob' can find a certificate for an incorrect y' on which Alice will accept, except with negligible probability. Clearly, Alice must take the time to send Π to Bob and to read y. W.l.o.g. we assume $|y| = |\Pi| < t$. However, if $|y| << t$, the question of fast verification of y becomes important. We show that Alice can do so in an efficient manner:

THEOREM 3 Assume that one-round $PIR(cc, db, u)$ scheme exist. Then, there exists correct verification of any remote procedure call (Π, t) such that the running time of Bob is $O(t^{O(1)} + db(t^{O(1)}, k))$; the size of public key, private key and the certificate are $O(\log^{O(1)} t \cdot cc(t^{O(1)}, k))$, and the running time of Alice is $O(|y| + \log^{O(1)} t \cdot u(t^{O(1)}, k))$.

We stress that theorem 3 holds for *any* one-round implementation of PIR protocol. For example, if we use Cachin at al. implementation of PIR, based on ϕ-hiding assumption [6], and $k(t) = \log^{O(1)} t$ for sufficiently large t, then we achieve poly-logarithmic bounds for the verification of y. We again remark that using *holographic proofs* [3,22], we can make the running time of the prover in the above theorem nearly-linear. Combining, we achieve:

COROLLARY 3 Assume that ϕ-hiding assumption holds. Then, for any $\epsilon > 0$, there exists correct verification of any remote procedure call (Π, t) such that for t sufficiently large, the running time of Bob is $O(t^{1+\epsilon})$; the size of public key, private key and the certificate are all $O(\log^{O(1)} t)$, and the running time of Alice is $O(|y| + \log^{O(1)} t)$.

We remark that if Π is a probabilistic computation, the above theorems also hold. Moreover, the coin-flips of Bob can be made *witness-indistinguishable*, similar to theorem 2.

4 Constructions

First, we describe our construction for theorem 1. Recall that the Prover is given x and a witness w. Let $|PCP(x, w)| = N$ be Håstad's 3-query PCP proof that $x \in L$ [14]. We remark that the use of of Hastad's version of PCP is not essential in our construction, and we can replace it with any *holographic proofs* [3,22] that achieves negligible error probability, and thus we can improve improve the running time of the prover to be nearly-linear. We choose to use Hastad's version of PCP since is somewhat simplifies the presentation of our main theorem. However, we stress that *permuting* PCP queries is essential in our proof.

In the construction of theorem 3 simply note that Prover can write down the *trace* of the execution of the procedure call, which serves as a *witness* that the output is correct, and the same construction applies. The corollaries 2, 1, and 3 follow from our construction by simply by plugging in the appropriate implementation of PIR protocols.

V: For $j = 1, \ldots, \log^2 N$ do:
- Choose $(i_1, i_2, i_3)_j \in [1, N]$ according to Hastad's PCP Verifier.
- Choose a random permutation σ_j over $\{i_1, i_2, i_3\}$. Let $(i'_1, i'_2, i'_3)_j = \sigma_j(i_1, i_2, i_3)$.
- Compute 3 *independent* PIR encodings of i'_1, i'_2, i'_3 $PIR_j(i'_1)$ $PIR_j(i'_2)$ $PIR_j(i'_3)$ (i.e. 3 queries for retrieval from N-bit database, each having its own independent encoding and decoding keys)

$V \to P$: For $1 \leq j \leq \log^2$ send $PIR_j(i'_1)$ $PIR_j(i'_2)$ $PIR_j(i'_3)$

$P \to V$: Prover computes Hastad PCP proof on (x, w) treats it as an N-bit database, evaluates $3 \log^2 n$ PIR queries received from the verifier and sends the PIR answers back to the Verifier.

V: For $1 \leq j \leq \log^2 n$ PIR-decode the answers to $PIR_j(i'_1)$, $PIR_j(i'_2)$, $PIR_j(i'_3)$, apply σ_j^{-1} to get the answers to $(i_1, i_2, i_3)_j$. If for all $1 \leq j \leq \log^2 n$, Hastad's PCP verifier accepts on answers to PCP queries $(i_1, i_2, i_3)_j$ then accept, else reject.

Next, we describe the construction of theorem 2, which also achieves witness-indistinguishability. We shall use Zero-Knowledge PCP and one-round SPIR. Again, we denote $|ZKPCP(w, x)| = N$ (recall that N is polynomial in $|w|$). This time by j we denote $O(\log^{O(1)} N)$ queries needed by ZKPCP to achieve negligible error probability while maintaining super-logarithmic bound on the number of bits needed to be read by the ZKPCP verifier to break the Zero-Knowledge property of the PCP [17].

V: Choose $i_1, \ldots, i_j \in [1, N]$ according to ZKPCP Verifier; Choose a random permutation σ over $\{i_1, \ldots, i_j\}$; Let $(i'_1, \ldots, i'_j) = \sigma(i_1, \ldots, i_j)$; Compute j *independent* SPIR encodings $SPIR_1(i'_1), \ldots, SPIR_j(i;_j)$ (i.e. j queries for retrieval from N-bit database, each having its own independent PIR encoding and decoding keys);

$V \to P$: Send $SPIR_1(i'_1), \ldots, SPIR_j(i'_j))$.

$P \to V$: Prover computes ZKPCP proof on (x, w) and treats it as an N-bit database, evaluates j received SPIR queries on it and sends SPIR answers back to the Verifier.

V: Decode j SPIR answers to $SPIR_1(i'_1), \ldots, SPIR_j(i'_j))$, apply σ^{-1} to get the answers to $i_1, \ldots, i_j$ queries and check if ZKPCP verifier accepts on answers to $i_1, \ldots, i_j$. If so accept, else reject.

5 Proof of Theorem 1

In order to prove Theorem 1, we need to prove completeness and soundness. We will prove them in that order.

The proof of completeness follows from our construction, i.e. there exist algorithms for C and (honest) S such that for every triple f, x, y where f is a polynomial time computation, there is a certificate which is the set of correct answers for every query set from a PCP verifier. The PIR encoding of this query set can be efficiently computed by C. Similarly, the correct PCP answers to the query set and their PIR encoding can also be computed efficiently by S. There are $\log^{O(1)} n$ PIR queries each of which can be encoded in the [6] construction using $\log^{O(1)} n$ bits giving a total communication complexity of $\log^{O(1)} n$ bits both in the query and the response.

The intuition of the soundness proof is that even though a cheating server may use a different proof string for each query response, we can construct an "emulator" algorithm for this server that flips its coins independent of the server and the PIR encoded queries, selects one "proof" string according to some probability distribution, and responds to queries honestly according to this string. Most of the work in the proof will be in showing that the induced distribution on the query responses from the emulator is close to that of the server and hence the error bounds of the PCP theorem apply with some small slack. Recall that N is the length of the PCP proof string and let $[N]$ represent $\{1, 2, \ldots, N\}$. ϵ_{PCP} is the acceptance error of the PCP verifier using l queries and ϵ_{PIR} is the error probability of the PIR scheme used. Let $I_1, I_2, \ldots, I_l$ be random variables representing the l queries over $[N]$ asked by the verifier V and $B_1, \ldots, B_l$ be the random variables representing the decoded bit responses of the server. Let $P^{i_1 \ldots i_l}_{b_1 \ldots b_l} = \Pr[B_1 = b_1 \ldots B_l = b_l | I_1 = i_1, \ldots, I_l = i_l]$ be the probability distribution of the server's responses to queries where the choice is over the server's coin flips and PIR-encodings.

Given the distribution $P^{i_1 \ldots i_l}_{b_1 \ldots b_l}$, we will construct an emulator algorithm which probabilistically (using its own coin flips and independent of the server and queries) chooses a proof string and answers the queries according to this string. Furthermore, the distribution induced on the emulator's responses by this choice of string will be close to $P^{i_1 \ldots i_l}_{b_1 \ldots b_l}$. We will show that the PCP error bound applies to the emulator's responses and thence that the error bound will also apply to the server's responses but with some additional slack. Let the emulator have a probability distribution Q on N-bit strings and let $\tilde{B}_1 \ldots \tilde{B}_l$ be random variables representing the bits of the proof string chosen according to Q. Q induces a probability distribution on the emulator's responses to queries. Denote this induced distribution by $\tilde{P}^{i_1 \ldots i_l}_{b_1 \ldots b_l} = \Pr[\tilde{B}_1 = b_1 \ldots \tilde{B}_l = b_l | I_1 = i_1, \ldots, I_l = i_l]$. Define $\epsilon_S := \max_{i_1 \ldots i_l, b_1 \ldots b_l} |P^{i_1 \ldots i_l}_{b_1 \ldots b_l} - \tilde{P}^{i_1 \ldots i_l}_{b_1 \ldots b_l}|$. First, we note that the emulator's probability of acceptance by the PCP verifier is bounded by ϵ_{PCP}.

CLAIM 1 $\Pr[V(I_1, \ldots, I_l; \tilde{B}_1 \ldots \tilde{B}_l) = 1] \leq \epsilon_{PCP}$.

Proof: $\Pr[V(I_1, \ldots, I_l; \tilde{B}_1 \ldots \tilde{B}_l) = 1] = \sum_{D \in \{0,1\}^N} \Pr[V(I_1, \ldots, I_l; d_{I_1} \ldots_{I_l}) = 1 | D = d] \Pr[D = d]$ where d_{I_j} denotes the I_j-th bit of d. From the PCP Theorem, for all d, $\Pr[V(I_1, \ldots, I_l; d_{I_1} \ldots_{I_l}) = 1] \leq \epsilon_{PCP}$. Applying this to the previous equation we get, $\Pr[V(I_1, \ldots, I_l; \tilde{B}_1 \ldots \tilde{B}_l) = 1] \leq \sum_{D \in \{0,1\}^N} \epsilon_{PCP} \Pr[D = d] = \epsilon_{PCP}$. □

Next, we note that the server's probability of acceptance by the Verifier is close to that of the emulator.

LEMMA 1 $\Pr[V(I_1, \ldots, I_l; B_1 \ldots B_l) = 1] \leq \epsilon_{PCP} + 2^l \epsilon_S$.

Proof: From the PCP Theorem, $\Pr[V(I_1, \ldots, I_l; \tilde{B}_1 \ldots \tilde{B}_l) = 1] \leq \epsilon_{PCP}$. Let $i_1, \ldots, i_l \in [N]$ be query instances and $b_1 \ldots b_l$ be bit strings. Using conditional probabilities we can express the LHS as a sum over all query instances and response bit strings. Using $|\tilde{P}^{i_1 \ldots i_l}_{b_1 \ldots b_l} - P^{i_1 \ldots i_l}_{b_1 \ldots b_l}| \leq \epsilon_S$ and collapsing the sums we get, $\Pr[V(I_1, \ldots, I_l; B_1 \ldots B_l) = 1] \leq \Pr[V(I_1, \ldots, I_l; \tilde{B}_1 \ldots \tilde{B}_l) = 1] + \epsilon_S \sum_{I_1 \ldots I_l} \sum_{b_1 \ldots b_l} \leq \epsilon_{PCP} + 2^l \epsilon_S$. □

Next, we show the existence of an emulator whose response distribution is close to the server's. Note mere proof of existence suffices.

LEMMA 2 There exists an "emulator" algorithm such that $\epsilon_S \leq \epsilon_{PIR} \cdot 2^{O(l \log l)}$.

For clarity of exposition, we write the proof using only 3-tuple queries and provide the calculation for $l > 3$ queries at the end. First, we make a simple claim which follows from the fact that the client chooses a random permutation of the 3-query, hence the server strategy has to be oblivious of the order of elements within the 3-tuple. Thus, w.l.o.g. we only need to consider the distributions $P^{i,j,k}$ where $1 \leq i < j < k \leq N$.

CLAIM 2 For all permutations σ on $\{1,2,3\}$ and all 3-tuples i_1, i_2, i_3, and all decodings b_1, b_2, b_3, $P^{i_1,i_2,i_3}_{b_1,b_2,b_3} = P^{i_{\sigma(1)},i_{\sigma(2)},i_{\sigma(3)}}_{b_{\sigma(1)},b_{\sigma(2)},b_{\sigma(3)}}$.

We want an emulator whose probability distribution Q over N-bit strings is consistent with the distribution of the server's responses. Formally, if $Q^{[N]|i_1i_2i_3}_{b_1b_2b_3}$ is the probability distribution induced by Q on indices i_1, i_2, i_3, then we want $\forall i_1, i_2, i_3, b_1, b_2, b_3$ $Q^{[N]|i_1,i_2,i_3}_{b_1b_2b_3} = P^{i_1i_2i_3}_{b_1b_2b_3}$ to be true. What we will actually show that there is a emulator for which this is true with small error, i.e. $max_{i_1i_2i_3,b_1b_2b_3}|Q^{[N]|i_1i_2i_3}_{b_1b_2b_3} - P^{i_1i_2i_3}_{b_1b_2b_3}| \leq \epsilon_S$. To show the existence of such an emulator, we write out the equations that the above equality implies on the actual proof strings that Q is over.

Construct matrix A with $2^3\binom{N}{3}$ rows and 2^N columns as follows: the rows of A are labeled by the different query-response tuples $r^{ijk}_{b_1,b_2,b_3}$ and the columns are labeled by the 2^N possible N-bit strings. Let B be the vector of $8\binom{N}{3}$ of values $P^{i,j,k}_{b_1,b_2,b_3}$ from the server's strategy. Let x be the probability vector with 2^N entries where x_i is the probability that the i-th N bit string is chosen as the database. To prove Lemma 2, it is enough to show that the system $Ax = B$ can be solved for the probability vector x. Any such solution can be used for the distribution Q.

For clarity, we label the $8\binom{N}{3}$ rows of matrix A in lexicographic order over the tuples and in increasing order of weight over bit strings $r^{123}_{000}, r^{123}_{001}, r^{123}_{010}, r^{123}_{100}, r^{123}_{011} \ldots r^{N-2,N-1,N}_{111}$ and the 2^N columns of matrix A by subsets of $[N]$ enumerated in lexicographic order. That is, let $\emptyset, \{1\}, \{2\}, \ldots, \{N\}, \{1,2\}, \{1,3\}, \ldots, \{N-1,N\}, \ldots, \{1,\ldots,N\}$ represent the labels of the columns of A in order. Now we can describe how the elements of A can be filled: let $I = r^{i,j,k}_{b_1b_2b_3}$ be the label of the row in question and let $R^{i,j,k}_{b_1b_2b_3}$ be the actual vector. Let $J \subset \{1,\ldots,N\}$ be the label of the column, then set $A[I,J] = 1$ if $D_J[I] = 0$, i.e. if the J-th string has a zero in its I-th position. Clearly, if there is a solution x which is a probability distribution on N-bit strings, then the emulator using this distribution can generate the same distribution as the server on query-response tuples.

In order to prove that a solution exists, we now define a series of row transformations on A in lexicographic order. For the first row R^{123}_{000} there is no change. For the next row, r^{123}_{001}, we add the previous row to this one. That is, the second row with label r^{123}_{001} is now $R^{123}_{000} + R^{123}_{001}$. Define $R^{i,j}_{0,0}$ as the N bit vector which is defined as: $R^{i,j}_{0,0}[l] = 1$ if the l-th column label contains i and j. Note that $R^{i,j,k}_{0.0.0}$ and $R^{i,j,k}_{0,0,1}$ can not both be 1 in any index. Hence, the row with label r^{123}_{001} has the row R^{12}_{00}. Next, for the row with label r^{123}_{010}, we add the row labeled r^{123}_{000} to get the row R^{13}_{00} and similarly, for the row labeled r^{123}_{100}. Finally, for the rows labeled r^{123}_{011} we add the rows labeled r^{123}_{000}, r^{123}_{001} and r^{123}_{010}. Defining R^i_0 as the vector with 1 in all the positions where the i-th bit is 1, we note that the row labeled r^{123}_{011} has the row R^1_0. Analogously, we transform the rows labeled r^{123}_{101} to R^2_0 and r^{123}_{110} to R^3_0. Finally, the row labeled r^{123}_{111} can be transformed to the all 1's vector by adding to it all the previous rows. Next in the lexicographic order are the rows labeled r^{124}_{000} through r^{124}_{111}. We follow the same procedure outlined above for these rows to get the rows $R^{124}_{000}, \ldots, R^4_0, \mathbf{1}$. Since every one of the $8\binom{N}{3}$ rows in the matrix has a label of one of these forms we can carry out this procedure for all the rows of A to give us a new matrix A'.

In order that the solutions to the system $Ax = B$ remain unchanged when we transformed A to A', we have to perform the same set of row operations on B. To start with, consider B^{123}_{001}. To be consistent with A, we add B^{123}_{001} to give $B^{123|12}_{00}$. Similarly, we add B^{123}_{000} to B^{123}_{010}. Let this sum be called $B^{123|13}_{00}$ and analogously for $B^{123|23}_{00}$. Next, for B^{123}_{011}, we add $B^{123}_{000}, B^{123}_{001}$ and B^{123}_{010} to get $B^{123|1}_0$. Similarly, we get $B^{123|2}_0$ and $B^{123|3}_0$. Finally, replace B^{123}_{111} by the sum of all the eight quantities i.e. $\sum_{b_1b_2b_3} B^{123}_{b_1b_2b_3}$. Follow this procedure for all the values of B to give a new vector B'.

CLAIM 3 The systems $Ax = B$ and $Ax' = B'$ have exactly the same set of solutions.

Proof: We can write the row transformations that we have performed as a matrix T that left multiplies both sides. Since we have performed the same transformations on B as on A, we get $TAx = TB$. Trivially, T is an invertible matrix. □

Now, we proceed to show that the system $A'x = B'$ is solvable , i.e. there is at least one solution which is a probability. As is obvious from the construction of A', all rows are duplicated many times. Hence A' is not full rank and we have to show that B' is in the column space of A'. We do this by collecting all the unique rows and reordering the rows of matrix A' so that it becomes upper triangular. Since, we want A' to be upper triangular, we must dovetail the row order to the column order that we have already established. Hence, we list the rows in the following order now: the first row will be the vector $R^{\emptyset}_{\emptyset}$ which is the all ones vector. Next, we list in order $R^i_0, i = 1, \ldots, N$ (only one copy each). Consider the elements of the row R^1_0. Recall that the columns of A' (as were the columns of A) are labeled as $\emptyset, \{1\}, \{2\}, \ldots, \{N\}, \{1, 2\}, \ldots$. Hence, R^1_0 is of the form $0, 1, \ldots$ whereas R^2_0 has the form $0, 0, 1, \ldots$ and so on. Next, we list in order one copy each of $R^{i,j}_{00}$. Similarly, $R^{1,2}_{0,0}$ will have $N + 1$ leading zeros and then a 1 in the column labeled $\{1, 2\}$. One can similarly write the elements of the rows of the form $R^{i,j,k}_{0,0,0}$. We have accounted for $l = 1 + \binom{N}{1} + \binom{N}{2} + \binom{N}{3}$ rows and by listing the rows in this order gives an upper triangular part of A'. To preserve equivalence, we have to reorder the elements of B' in the same way. The first element of B' is 1. The second element in B' (corresponding to row R^1_0) is $P^{123|1}_0$. Similarly, the next $N-1$ elements are $P^{123|2}_0$ through $P^{N-2,N-1,N|N}_0$. The next set of elements are $P^{123|12}_{00}$ through $P^{N-2,N-1,N|N-1,N}_{00}$ followed by P^{123}_{000} through $P^{N-2,N-1,N}_{000}$.

The remaining $8\binom{N}{3} - (1 + \binom{N}{1} + \binom{N}{2} + \binom{N}{3})$ rows are all duplicates of the rows in the upper triangle since our transformation from A to A' transformed all the rows. For example, the row R^{ij}_{00} will be repeated $N - 2$ times, once for every $k \in [N], k \neq i \neq j$. Similarly, R^i_0 will be repeated once for every pair $j, k \in [N], i \neq j \neq k$. This gives us an easy construction of the left null space $L_{A'}$ of A'. For every i-th row not in the upper triangle, we know that it is a duplicate of some j-th row in the upper triangle. We can construct a vector in $L_{A'}$ which has a 1 in the i-th position and -1 in the j-th position. This construction describes all the vectors in $L_{A'}$. Now, it is sufficient that B' be orthogonal to all the vectors in $L_{A'}$, i.e. $\forall y \in L_{A'}(y, B') = 0$. By our construction, every such y has a -1 in one of the first t elements and a $+1$ in one of the remaining positions. This gives us the following equality constraints between elements of B'. The duplicates of the rows in A' of the form $R^{i,j}_{00}$ give constraints of the form:

$$\forall i, j, k, k' \in [N], i \neq j \neq k \neq k', P^{ijk}_{000} + P^{ijk}_{001} = P^{ijk'}_{000} + P^{ijk'}_{001}.$$

All the duplicate rows in A' of the form R^i_0 give constraints of the form

$$\forall i, j, j', k, k' \in [N], i \neq j \neq j' \neq k \neq k'$$

$$P^{ijk}_{000} + P^{ijk}_{001} + P^{ijk}_{010} + P^{ijk}_{011} = P^{ij'k'}_{000} + P^{ij'k'}_{001} + P^{ij'k'}_{010} + P^{ij'k'}_{011}.$$

Hence, if these constraints are true on B' then B' is in the column space of A'.

Define projections of the probability distributions $P^{i_1 i_2 i_3}$ as follows. Let $P^{i_1 i_2 i_3 | i_1 i_2}_{b_1 b_2} = P^{i_1 i_2 i_3}_{b_1 b_2 0} + P^{i_1 i_2 i_3}_{b_1 b_2 1}$. Similarly, define projections of the form $P^{i_1}_0$. The following claim follows from the assumption of zero error probability for PIR. It says that the server's response on a particular query index cannot depend on the tuple.

CLAIM 4 Let t be any single query or two query indices and let s and s' be any three-query tuples containing t. Then, if $\epsilon_{PIR} = 0$, $P^{s|t} = P^{s'|t}$.

By the claim above, projection probabilities are well defined. Let $P^{i_1,i_2} := P^{s|i_1,i_2}$ for any s which contains i_1 and i_2. Likewise, P^{i_1} is defined analogously.

LEMMA 3 The system $A'x = B'$ yields a valid set x of probabilities on N-bit strings when $\epsilon_{PIR} = 0$.

Proof: We have shown above that the system $A'x = B'$ has a solution when B' satisfies the projection constraints above. Indeed, that these constraints are satisfied follows immediately from the assumption of zero-error PIR. However, we still have to ensure that the solutions we have are probabilities. The first row ensures that all the elements of the solution vector sum up to 1. However, it is not obvious that the elements are non-negative. To see this, we break the solution to the upper triangular system into blocks. An (i,j) block is a subset of rows consisting of the row $R_{00}^{i,j}$ and all rows of the form R_{000}^{ijk}. Note that P_{000}^{ijk} is already non-negative since it is a given probability. The constraints on P_{00}^{ij} imply that $P_{00}^{ij} = P_{000}^{ijk} + P_{001}^{ijk}$ for all $k \neq i \neq j$. Furthermore, the rows R_{00}^{ij} has no common 1 positions with any row except rows of the form R_{00*}^{ijk}. From this we can infer that a non-negative solution exists in this block. We can follow the same argument for all such (i,j) blocks since they are independent. Finally, by an analogous argument, it can be shown that the i blocks can be solved with non-negative solutions as well. □

It now remains to consider the case of $\epsilon_{PIR} > 0$. First, we note that permutations of a query do not give any advantage to the adversary since we chose a random permutation of any query. However, the projection constraints may not hold exactly, and hence existence of a solution to $A'x = B'$ is not a given. A PIR scheme with $\epsilon_{PIR} > 0$ implies that the projection constraints are satisfied with error bounded by ϵ_{PIR}. That is, for example, $\forall i \neq j \neq k \neq k' \in [N]$ $|P_{000}^{ijk} + P_{001}^{ijk} - (P_{000}^{ijk'} + P_{001}^{ijk'})| \leq \epsilon_{PIR}$. The following lemma shows that there is a solution $\tilde{P}$ that satisfies all the projection constraints and is close to the given probabilities P.

LEMMA 4 There exists a probability vector $\tilde{P}_{b_1 \ldots b_l}^{i_1 \ldots i_l}$ such that the system $Ax = \tilde{P}$ is solvable and $max_{i_1 \ldots i_l, b_1 \ldots b_l} |P_{b_i b_j b_k}^{ijk} - \tilde{P}_{b_i b_j b_k}^{ijk}| \leq 2^{O(l \log l)} \epsilon_{PIR}$.

Proof: To create the vector $\tilde{P}$ of probabilities we first compute $\bar{P}_{00}^{ij} := \frac{1}{N-2} \sum_{k \neq i \neq j} \tilde{P}_{00}^{ijk|ij}$ where $\tilde{P}_{b_i b_j}^{ijk|k} := \tilde{P}_{b_i b_j 0}^{ijk|k} + \tilde{P}_{b_i b_j 1}^{ijk|k}$. Similarly, $\bar{P}_0^i := \binom{N-1}{2}^{-1} \sum_{k \neq i \neq j} \tilde{P}_0^{ijk|i}$ with $\tilde{P}_0^{ijk|i}$ defined analogously. We now show how to compute $\tilde{P}_{b_i b_j b_k}^{ijk}$ from these values. There are 8 values to be calculated $\tilde{P}_{000}^{ijk}$ through P_{111}^{ijk}. Start with $P_{000}^{ijk} = P_{000}^{ijk}$. Given the values $\bar{P}_{00}^{ij}$ and $\bar{P}_0^i$ we can compute all the $\tilde{P}_{b_i b_j b_k}^{ijk}$ such that they satisfy the projection constraints exactly as follows. $\tilde{P}_{001}^{ijk} := \bar{P}_{00}^{ij} - \tilde{P}_{000}^{ijk}$. Similarly, we can compute $\tilde{P}_{010}^{ijk} := \bar{P}_{00}^{ik} - \tilde{P}_{000}^{ijk}$ and $\tilde{P}_{100}^{ijk} := \bar{P}_{00}^{jk} - \tilde{P}_{000}^{ijk}$. Then, $\tilde{P}_{011}^{ijk} := \bar{P}_0^i - (\tilde{P}_{000}^{ijk} + \tilde{P}_{001}^{ijk} + \tilde{P}_{010}^{ijk})$ and so on. Finally, we have to show that the probabilities $\tilde{P}$ are within some ϵ of P. We can compute the distance by tracing the computation path of these probabilities above. The error bound depends on the number of queries l asked and so we state this last bound in terms of l.

CLAIM 5 For any client making l queries, there exists a probability vector $\tilde{P}$ satisfying the projection constraints such that for all $i_1, \ldots, i_l$ and $b_1 \ldots b_l$, $|P_{b_1 \ldots b_l}^{i_1 \ldots i_l} - \tilde{P}_{b_1 \ldots b_l}^{i_1 \ldots i_l}| \leq 2^{O(l \log l)} \epsilon_{PIR}$.

This claim completes the proof of Lemma 2. We omit the simple proof for lack of space. Proof of Theorem 1: Given a cheating server whose acceptance probability for a language $L \in NP$ is $\geq \epsilon_{PCP}$, using Lemmas 1 and 2, there is an emulator whose acceptance probability is $\geq \epsilon_{PCP} + \epsilon_{PIR} \cdot 2^{O(l \log l)}$. ϵ_{PCP} is already negligible by assumption. The second term can be made negligible by increasing the security parameter in the PIR scheme by a poly-logarithmic factor. □

6 Extensions and Further Results

We briefly sketch the proof of Theorem 2. It is straightforward to see that Theorem 2 carries all the properties of Theorem 1 using essentially the same proof (i.e. showing that one can construct probability distribution on databases which will pass PCP theorem) but converting the PCP proof into a ZKPCP proof and using PIR instead of SPIR. To prove witness-indistinguishability, we now assume that the Prover is honest, and that there is a poly-bounded Verifier who wants to distinguish witness w_1 from witness w_2. SPIR guarantees that the verifier will not be able to read more than allowed by our protocol $\log^{O(1)} N$ bits and by the properties of ZKPCP the bits retrieved are computationally indistinguishable for w_1 and w_2. Hence, if the Verifier can distinguish this violates either SPIR security or the zero-knowledge property of ZKPCP [17]. This sketch will be elaborated in the full version of the paper.

The proof of Theorem 1 also has implications for one-round multi-prover proof systems in the information-theoretic setting. Notice that we show in Theorem 1 how to combine PIR protocols with PCP in one round. In the multi-prover setting, information-theoretic implementations of PIR are known [5,1] with small communication complexity. Our theorem provides an alternative way to reduce the communication complexity in the multi-prover setting: all provers can write down a PCP proof on their work tapes, and then the communication complexity is simply the cost of retrieving poly-logarithmic number of bits using information-theoretically secure PIR protocols. So far, the best bounds known for retrieving a single bit using info-theoretic PIR with only two provers is $O(N^{\frac{1}{3}})$ [5]. Thus, our approach gives an inferior result to [10]. However, our technique works with any multi-database PIR protocol and thus an improvement in information-theoretic PIR protocols would improve the communication complexity in our approach. We remark as an aside that our results also hold in the setting of universal service provider PIR [8].

Note that our "emulation" technique is not PCP-specific. More specifically, any adversary who does not answer PIR queries honestly can be emulated (with small error) by an algorithm that flips coins independently of the PIR encodings to choose a database which it then uses to answer the queries honestly.

We also wish to point out that the same Verifier's message can be used for multiple proofs to multiple provers, and they *all* can be checked for correctness (with the same error probability) without any additional messages from the verifier. As long as the Verifier does not reveal any additional information about his private key or whether he accepted or rejected each individual input, the error probability holds.

Acknowledgements

We wish to thank Silvio Micali and Yuval Ishai for helpful discussions.

References

1. A. Ambainis. Upper bound on the communication complexity of private information retrieval. In *Proc. of 24th ICALP*, 1997.
2. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.
3. L. Babai, L.Fortnow, L. Levin and M. Szegedy Checking computations in polylogarithmic time. In Proc. of the *23rd Annual ACM Symposium on the Theory of Computing*, pp. 21-31, 1991.
4. I. Biehl and B. Meyer and S. Wetzel. Ensuring the Integrity of Agent-Based Computation by Short Proofs. Mobile Agents '98, Kurt Rothermel and Fritz Hohl, editors, Lecture Notes in Computer Science, Vol. 1477, 1998, Springer-Verlag, pp. 183–194.
5. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Proc. of 36th FOCS*, pages 41–50, 1995. Journal version to appear in JACM.

6. C. Cachin, S. Micali, and M. Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology: Proceedings of EUROCRYPT99*, 1999.
7. C. Dwork and M. Naor Zaps and Apps talk given at DIMACS Workshop on Cryptography and Intractability March 20 - 22, 2000 DIMACS Center, Rutgers University, Piscataway, NJ.
8. G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Universal service-providers for database private information retrieval. In *Proc. of the 17th Annu. ACM Symp. on Principles of Distributed Computing*, pages 91-100, 1998.
9. G. Di Crescenzo, T. Malkin, and R. Ostrovsky. Single-database private information retrieval implies oblivious transfer. In *Advances in Cryptology - EUROCRYPT 2000*, 2000.
10. U. Feige and J. Kilian, Two Prover Protocols: low error at affordable rates. In *Proc of Annual ACM Symposium on the Theory of Computing, 1994* pp. 172-183.
11. U. Feige and A. Shamir, Witness Indistinguishable and Witness Hiding Protocols. In *Proc. of Annual ACM Symposium on the Theory of Computing 1990*, pages 416-426.
12. F. Ergün, S Kumar, and R. Rubinfeld. Fast pcps for approximations. FOCS 1999.
13. Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin Protecting data privacy in private information retrieval schemes. In Proc. of the *30th Annual ACM Symposium on the Theory of Computing*, pp. 151-160, 1998.
14. J. Håstad. Some optimal inapproximability results. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 1–10, El Paso, Texas, 4–6 May 1997.
15. J. Kilian Zero-Knowledge with Logspace Verifiers. In *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*, pages 25-35, El Paso, Texas, 4–6 May 1997.
16. J. Kilian Improved Efficient Arguments In *Proc. of CRYPTO* Springer LNCS 963:311-324, 1995
17. J. Kilian, E. Petrank, and G. Tardos. Probabilistic Checkable Proofs with Zero Knowledge. In 29th ACM Symp. on Theory of Computation, May 1997.
18. E. Kushilevitz and R. Ostrovsky Replication is not needed: Single Database, computationally-private information retrieval. In *Proc. of Thirty-Eight Foundations of Computer Science* pp. 364-373, 1997.
19. E. Kushilevitz and R. Ostrovsky. One-way Trapdoor Permutations are Sufficient for Non-Trivial Single-Database Computationally-Private Information Retrieval. In *Proc. of EUROCRYPT '00*, 2000.
20. S. Micali. CS proofs (extended abstracts). In *35th Annual Symposium on Foundations of Computer Science*, pages 436–453, Santa Fe, New Mexico, 20–22 1994. IEEE.
21. M. Naor and B. Pinkas, Oblivious transfer and polynomial evaluation. In Proceedings of the 31st Annual Symposium on Theory of Computing. ACM, pp. 33-43, 1999.
22. A. Polishchuk and D. Spielman Nearly-linear Size Holographic Proofs In Proceedings of the 25st Annual Symposium on Theory of Computing. ACM, pp. 194-203, 1994.
23. R. Raz A Parallel Repetition Theorem SIAM J. Computing Vol. 27, No. 3, pp. 763-803, June 1998.
24. B.S. Yee. A Sanctuary For Mobile Agents. Proceedings of the DARPA Workshop on Foundations for Secure Mobile Code. March, 1997.
25. A.C. Yao. Theory and Applications of Trapdoor Functions *In 23rd Annual Symposium on Foundations of Computer Science* pages 80-91, Chicago, Illinois, 3-5 November 1982. IEEE.

A New Unfolding Approach to LTL Model Checking *

Javier Esparza[1] and Keijo Heljanko[2]

[1] Institut für Informatik, Technische Universität München, Germany
esparza@in.tum.de
[2] Lab. for Theoretical Computer Science, Helsinki University of Technology, Finland
Keijo.Heljanko@hut.fi

Abstract. A new unfolding approach to LTL model checking is presented, in which the model checking problem can be solved by direct inspection of a certain finite prefix. The techniques presented so far required to run an elaborate algorithm on the prefix.

1 Introduction

Unfoldings are a partial order technique for the verification of concurrent and distributed systems, initially introduced by McMillan [11]. They can be understood as the extension to communicating automata of the well-known unfolding of a finite automaton into a (possibly infinite) tree. The unfolding technique can be applied to systems modelled by Petri nets, communicating automata, or process algebras [4,3,10]. It has been used to verify properties of circuits, telecommunication systems, distributed algorithms, and manufacturing systems [1].

Unfoldings have proved to be very suitable for deadlock detection and invariant checking [11]. For these problems, one first constructs a so-called *complete prefix* [4], a finite initial part of the unfolding containing all the reachable states. This prefix is at most as large as the state space, and usually much smaller (often exponentially smaller). Once the prefix has been constructed, the deadlock detection problem can be easily reduced to a graph problem [11], an integer linear programming problem [12], or to a logic programming problem [8].

In [2,7] and [17,16], unfolding-based model checking algorithms have been proposed for a simple branching-time logic and for LTL, respectively. Although the algorithms have been applied with success to a variety of examples, they are not completely satisfactory: After constructing the complete prefix, the model checking problem cannot be yet reduced to a simple problem like, say, finding cycles in a graph. In the case of LTL the intuitive reason is that the infinite sequences of the system are "hidden" in the finite prefix in a complicated way. In order to make them "visible", a certain graph has to be constructed. Unfortunately, the graph can be exponentially larger than the complete prefix itself.

* Work partially supported by the Teilprojekt A3 SAM of the Sonderforschungsbereich 342 "Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen", the Academy of Finland (Project 47754), and the Nokia Foundation.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 475–486, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Niebert has observed [13] that this exponential blow-up already appears in a system of n independent processes, each of them consisting of an endless loop with one single action as body. The complete prefix has size $\mathcal{O}(n)$, which in principle should lead to large savings in time and space with respect to an interleaving approach, but the graph i s of size $\mathcal{O}(2^n)$, i.e. as large as the state space itself.

In this paper we present a different unfolding technique which overcomes this problem. Instead of unrolling the system until a complete prefix has been generated, we "keep on unrolling" for a while, and stop when certain conditions are met. There are two advantages: (i) the model checking problem can be solved by a direct inspection of the prefix, and so we avoid the construction of the possibly exponential graph; and, (ii) the algorithm for the construction of the new prefix is similar to the old algorithm for the complete prefix; only the definition of a cut-off event needs to be changed. The only disadvantage is the larger size of the new prefix. Fortunately, we are able to provide a bound: the prefix of a system with K reachable states contains at most $\mathcal{O}(K^2)$ events, assuming that the system is presented as a 1-safe Petri net or as a product of automata[1]. Notice that this is an upper bound: the new prefix is usually much smaller than the state space, and in particular for Niebert's example it grows linearly in n.

The paper is structured as follows (for detailed definitions and proofs see the full version [5]). Section 2 presents the automata theoretic approach to LTL model checking. In Sect. 3 the unfolding method is introduced. Sections 4 and 5 contain the tableau systems for the two subproblems. In Sect. 6 we show how LTL model checking can be solved with the presented tableau systems. In Sect. 7 we conclude and discuss topics for further research.

2 Automata Theoretic Approach to Model Checking LTL

Petri nets. We assume that the reader is familiar with basic notions, such as net, preset, postset, marking, firing, firing sequence, and reachability graph. We consider labelled nets, in which places and transitions carry labels taken from a finite alphabet $\mathcal{L}$, and labelled net systems. We denote a labelled net system by $\Sigma = (P, T, F, l, M_0)$, where P and T are the sets of places and transitions, F is the flow function $F\colon (P \times T) \cup (T \times P) \to \{0, 1\}$, $l\colon P \cup T \to \mathcal{L}$ is the labelling function, and M_0 is the initial marking.

We present how to modify the automata theoretic approach to model checking LTL [15] to best suit the net unfolding approach. For technical convenience we use an action-based temporal logic instead of a state-based one, namely the linear temporal logic $tLTL'$ of Kaivola, which is immune to the stuttering of invisible actions [9]. With small modifications the approach can also handle state based stuttering invariant logics such as LTL-X. Given a finite set A of actions, and a set $V \subseteq A$ of visible actions, the abstract syntax of $tLTL'$ is given by:

$$\varphi ::= \top \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \mid \varphi_1 \,\mathcal{U}^a\, \varphi_2, \text{ where } a \in V$$

[1] More precisely, the number of non-cut-off events is at most $\mathcal{O}(K^2)$.

Formulas are interpreted over sequences of A^ω. The semantics of $\varphi_1 \mathcal{U} \varphi_2$ is as expected. Loosely speaking, a sequence w satisfies $\varphi_1 \mathcal{U}^a \varphi_2$ if φ_1 holds until the first a in w, and then φ_2 holds[2].

Given a net system $\Sigma = (P, T, F, l, M_0)$, where the transitions of T are labelled with actions from the set A, and a formula φ of $tLTL'$, the model checking problem consists of deciding if all the infinite firing sequences of Σ satisfy φ.

The *automata theoretic approach* attacks this problem as follows. First, a procedure similar to that of [6] converts the *negation* of φ into a Büchi automaton $\mathcal{A}_{\neg\varphi}$ over the alphabet $\Gamma = V \cup \{\tau\}$, where $\tau \notin A$ is a new label used to represent all the invisible actions. Then, this automaton is synchronized with Σ *on visible actions* (see [5] for details). The synchronization can be represented by a new labelled net system $\Sigma_{\neg\varphi}$ containing a transition (u, t) for every $u = q \xrightarrow{a} q'$ in $\mathcal{A}_{\neg\varphi}$ and for every $t \in T$, such that $l(t) = a$ and $a \in V$, plus other transitions for the invisible transitions of Σ. We say that (u, t) is an *infinite-trace monitor* if q' is a final state of $\mathcal{A}_{\neg\varphi}$, and a *livelock monitor* if the automaton $\mathcal{A}_{\neg\varphi}$ accepts an infinite sequence of invisible transitions (a *livelock*) with q' as initial state. The sets of infinite-trace and livelock monitors are denoted by I and L, respectively. An *illegal ω-trace* of $\Sigma_{\neg\varphi}$ is an infinite firing sequence $M_0 \xrightarrow{t_1 t_2 \ldots}$ such that $t_i \in I$ for infinitely many indices i. An *illegal livelock* of $\Sigma_{\neg\varphi}$ is an infinite firing sequence $M_0 \xrightarrow{t_1 t_2 \ldots t_i} M \xrightarrow{t_{i+1} t_{i+2} \ldots}$ such that $t_i \in L$, and $t_{i+k} \in (T \setminus V)$ for all $k \geq 1$. We have the following result:

Theorem 1. *Let Σ be a labelled net system, and φ a $tLTL'$-formula. $\Sigma \models \varphi$ if and only if $\Sigma_{\neg\varphi}$ has no illegal ω-traces and no illegal livelocks.*

The intuition behind this theorem is as follows. Assume that Σ can execute an infinite firing sequence corresponding to a word $w \in (V \cup \{\tau\})^\omega$ violating φ (where 'corresponding' means that the firing sequence executes the same visible actions in the same order, and an invisible action for each τ). If w contains infinitely many occurrences of visible actions, then $\Sigma_{\neg\varphi}$ contains an illegal ω-trace; if not, it contains an illegal livelock.

In the next sections we provide unfolding-based solutions to the problems of detecting illegal ω-traces and illegal livelocks. We solve the problems in an abstract setting. We fix a net system $\Sigma = (P, T, F, M_0)$, where T is divided into two sets V and $T \setminus V$ of *visible* and *invisible* transitions, respectively. Moreover, T contains two special subsets L and I. We assume that no reachable marking of Σ concurrently enables a transition of V and a transition of L. We further assume that M_0 does not put more than one token on any place. In particular, when applying the results to the model checking problem for $tLTL'$ and Petri nets, the system Σ is the synchronization $\Sigma_{\neg\varphi}$ of a Petri net and a Büchi automaton, and it satisfies these conditions. We use as running example the net system of Fig. 1. We have $V = \{t_6\}$, $I = \{t_1\}$, and $L = \{t_2\}$. The system has illegal ω-traces (for instance, $(t_1 t_3 t_4 t_6 t_7)^\omega$), but no illegal livelocks.

[2] Kaivola's semantics is interpreted over $A^* \cup A^\omega$, which is a small technical difference.

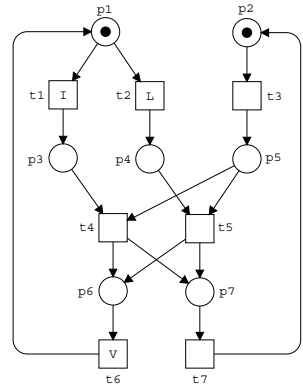


Fig. 1. A net system

3 Basic Definitions on Unfoldings

In this section we briefly introduce the definitions we need to describe the unfolding approach to our two problems. More details can be found in [4].

Occurrence nets. Given two nodes x and y of a net, we say that x is *causally related* to y, denoted by $x \leq y$, if there is a path of arrows from x to y. We say that x and y are in *conflict*, denoted by $x\#y$, if there is a place z, different from x and y, from which one can reach x and y, exiting z by different arrows. Finally, we say that x and y are *concurrent*, denoted by $x \mathit{co}\, y$, if neither $x \leq y$ nor $y \leq x$ nor $x\#y$ hold. A *co-set* is a set of nodes X such that $x \mathit{co}\, y$ for every $x, y \in X$. *Occurrence nets* are those satisfying the following three properties: the net, seen as a graph, has no cycles; every place has at most one input transition; and, no node is in self-conflict, i.e., $x\#x$ holds for no x. A place of an occurrence net is *minimal* if it has no input transitions. The net of Fig. 2 is an infinite occurrence net with minimal places a, b. The *default initial marking* of an occurrence net puts one token on each minimal place an none in the rest.

Branching processes. We associate to Σ a set of *labelled* occurrence nets, called the *branching processes* of Σ. To avoid confusions, we call the places and transitions of branching processes *conditions* and *events*, respectively. The conditions and events of branching processes are labelled with places and transitions of Σ, respectively. The conditions and events of the branching processes are subsets from two sets $\mathcal{B}$ and $\mathcal{E}$, inductively defined as the smallest sets satisfying:

- $\perp \in \mathcal{E}$, where $\perp$ is an special symbol;
- if $e \in \mathcal{E}$, then $(p, e) \in \mathcal{B}$ for every $p \in P$;
- if $\emptyset \subset X \subseteq \mathcal{B}$, then $(t, X) \in \mathcal{E}$ for every $t \in T$.

In our definitions we make consistent use of these names: The label of a condition (p, e) is p, and its unique input event is e. Conditions $(p, \perp)$ have no

input event, i.e., the special symbol $\perp$ is used for the minimal places of the occurrence net. Similarly, the label of an event (t, X) is t, and its set of input conditions is X. The advantage of this scheme is that a branching process is completely determined by its sets of conditions and events. We make use of this and represent a branching process as a pair (B, E).

Definition 1. *The set of* finite branching processes *of a net system Σ with the initial marking $M_0 = \{p_1, \ldots, p_n\}$ is inductively defined as follows:*

- $(\{(p_1, \perp), \ldots, (p_n, \perp)\}, \emptyset)$ *is a branching process of Σ.*[3]
- *If (B, E) is a branching process of Σ, $t \in T$, and $X \subseteq B$ is a co-set labelled by ${}^\bullet t$, then $(\, B \cup \{(p, e) \mid p \in t^\bullet\}\ ,\ E \cup \{e\}\,)$ is also a branching process of Σ, where $e = (t, X)$. If $e \notin E$, then e is called a* possible extension *of (B, E).*

The set of branching processes of Σ is obtained by declaring that the union of any finite or infinite set of branching processes is also a branching process, where union of branching processes is defined componentwise on conditions and events. Since branching processes are closed under union, there is a unique maximal branching process, called the *unfolding* of Σ. The unfolding of our running example is an infinite occurrence net. Figure 2 shows an initial part. Events and conditions have been assigned identificators that will be used in the examples. For instance, the event $(t_1, \{(p_1, \perp)\})$ is assigned the identificator 1.

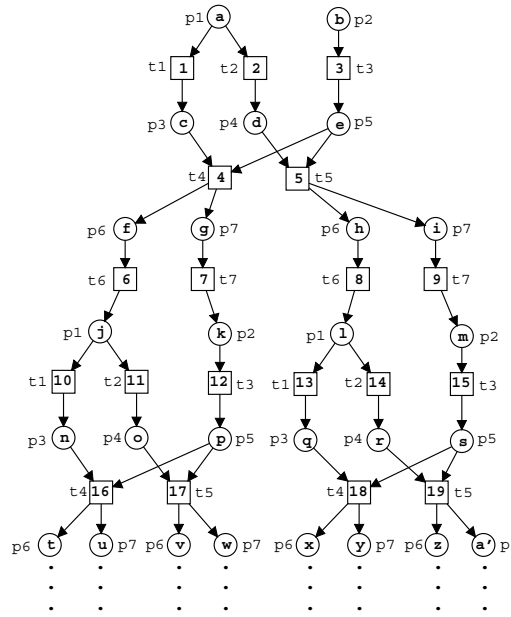


Fig. 2. The unfolding of Σ

[3] This is the point at which we use the fact that the initial marking is 1-safe.

Configurations. A *configuration* of an occurrence net is a set of events C satisfying the two following properties: C is causally closed, i.e., if $e \in C$ and $e' < e$ then $e' \in C$, and C is conflict-free, i.e., no two events of C are in conflict. Given an event e, we call $[e] = \{e' \in E \mid e' \leq e\}$ the *local configuration* of e. Let *Min* denote the set of minimal places of the branching process. A configuration C of the branching process is associated with a marking of Σ denoted by $Mark(C) = l((Min \cup C^\bullet) \setminus {}^\bullet C)$.

In Fig. 2, $\{1, 3, 4, 6\}$ is a configuration, and $\{1, 4\}$ (not causally closed) or $\{1, 2\}$ (not conflict-free) are not. A set of events is a configuration if and only if there is one or more firing sequences of the occurrence net (from the default initial marking) containing each event from the set exactly once, and no further events. These firing sequences are called *linearisations*. The configuration $\{1, 3, 4, 6\}$ has two linearisations, namely 1 3 4 6 and 3 1 4 6. All linearisations lead to the same reachable marking. For example, the two sequences above lead to the marking $\{p_1, p_7\}$. By applying the labelling function to a linearisation we obtain a firing sequence of Σ. Abusing of language, we also call this firing sequence a linearisation. In our example we obtain $t_1 t_3 t_4 t_6$ and $t_3 t_1 t_4 t_6$ as linearisations.

Given a configuration C, we denote by $\uparrow C$ the set of events $e \in E$, such that: (1) $e' < e$ for some event $e' \in C$, and (2) e is not in conflict with any event of C. Intuitively, $\uparrow C$ corresponds to the behavior of Σ from the marking reached after executing any of the linearisations of C. We call $\uparrow C$ the *continuation* after C of the unfolding of Σ. If C_1 and C_2 are two finite configurations leading to the same marking, i.e. $Mark(C_1) = M = Mark(C_2)$, then $\uparrow C_1$ and $\uparrow C_2$ are isomorphic, i.e., there is a bijection between them which preserves the labelling of events and the causal, conflict, and concurrency relations (see [4]).

4 A Tableau System for the Illegal ω-Trace Problem

In this section we present an unfolding technique for detecting illegal ω-traces. We introduce it using the terminology of tableau systems, the reason being that the technique has many similarities with tableau systems as used for instance in [18] for model-checking LTL, or in [14] for model-checking the mu-calculus. However, no previous knowledge of tableau systems is required.

Adequate orders. We need the notion of *adequate order* on configurations [4]. In fact, our tableau system will be parametric in the adequate order, i.e., we will obtain a different system for each adequate order. Given a configuration C of the unfolding of Σ, we denote by $C \oplus E$ the set $C \cup E$, under the condition that $C \cup E$ is a configuration satisfying $C \cap E = \emptyset$. We say that $C \oplus E$ is an *extension* of C. Now, let C_1 and C_2 be two finite configurations leading to the same marking. Then $\uparrow C_1$ and $\uparrow C_2$ are isomorphic. This isomorphism, say f, induces a mapping from the extensions of C_1 onto the extensions of C_2; the image of $C_1 \oplus E$ under this mapping is $C_2 \oplus f(E)$.

Definition 2. *A partial order $\prec$ on the finite configurations of the unfolding of a net system is an* adequate order *if:*

- *$\prec$ is well-founded,*
- *$C_1 \subset C_2$ implies $C_1 \prec C_2$, and*
- *$\prec$ is preserved by finite extensions; if $C_1 \prec C_2$ and $Mark(C_1) = Mark(C_2)$, then the isomorphism f from above satisfies $C_1 \oplus E \prec C_2 \oplus f(E)$ for all finite extensions $C_1 \oplus E$ of C_1.*

Total adequate orders are particularly good for our tableau systems because they lead to stronger conditions for an event to be a terminal, and so to smaller tableaux. Total adequate orders for 1-safe Petri nets and for synchronous products of transition systems, have been presented in [4,3].

4.1 The Tableau System

Given a configuration C of the unfolding of Σ, denote by $\#_I C$ the number of events $e \in C$ labelled by transitions of I.

Definition 3. *An event e of a branching process BP is a* repeat *(with respect to $\prec$) if BP contains another event e', called the* companion *of e, such that $Mark([e']) = Mark([e])$, and either*

(I) $e' < e$, or
(II) $\neg(e' < e)$, $[e'] \prec [e]$, and $\#_I[e'] \geq \#_I[e]$.

A terminal *is a minimal repeat with respect to the causal relation; in other words, a repeat e is a terminal if the unfolding of Σ contains no repeat $e' < e$. Repeats, and in particular terminals, are of type I or type II, according to the condition they satisfy.*

Events labelled by I-transitions are called I-events. A repeat e with companion e' is successful *if it is of type I, and $[e] \setminus [e']$ contains some I-event. Otherwise it is* unsuccessful.

A tableau *is a branching process BP such that for every possible extension e of BP at least one of the immediate causal predecessors of e is a terminal. A tableau is* successful *if at least one of its terminals is successful.*

Loosely speaking, a tableau is a branching process which cannot be extended without adding a causal successor to a terminal. In the case of a terminal of type I, $\uparrow[e]$ need not be constructed because $\uparrow[e']$, which is isomorphic to it, will be in the tableau. In the case of a terminal of type II, $\uparrow[e]$ need not be constructed either, because $\uparrow[e']$ will appear in the tableau. However, in order to guarantee completeness, we need the condition $\#_I[e'] \geq \#_I[e]$.

The tableau construction is straightforward. Given $\Sigma = (N, M_0)$, where $M_0 = \{p_1, \ldots, p_n\}$, start from the branching process $(\{(p_1, \bot), \ldots, (p_n, \bot)\}, \emptyset)$. Add events according to the inductive definition of branching process, but with the restriction that no event having a terminal as a causal predecessor is added. Events are added in $\prec$ order; more precisely, if $[e] \prec [e']$, then e is added before e'. The construction terminates when no further events can be added.

We construct the tableau corresponding to the net system of Fig. 1 using the total adequate order of [4].[4] All we need to know about this order is that for the events 4 and 5 in Fig. 2, $[4] \prec [5]$ holds. The tableau is the fragment of the unfolding of Fig. 2 having events 16, 17, and 5 as terminals. Events 16 and 17 are terminals of type I having event 4 as companion. Event 16 is successful because the set $[16] \setminus [4] = \{6, 7, 10, 11, 12, 16\}$ contains an I-event, namely 10. The intuition behind these terminals is rather clear: a terminal of type I corresponds to a cycle in the reachability graph. Loosely speaking, the events of $[16] \setminus [4]$ correspond to a firing sequence leading from $Mark([4])$ to $Mark([16])$, and these two markings coincide. Since $[16] \setminus [4]$ contains an I-event, the firing sequence contains a transition of I, and so we have found an illegal ω-trace. The set $[17] \setminus [4]$ doesn't contains any I-event, but $\uparrow[17]$ need not be constructed, because it is isomorphic to $\uparrow[4]$. Event 5 is a terminal of type II with event 4 as companion because $Mark([4]) = \{p_6, p_7\} = Mark([5])$, $[4] \prec [5]$, and $1 = \#_I[4] \geq \#_I[5] = 0$. The intuition is that $\uparrow[5]$ need not be constructed, because it is isomorphic to $\uparrow[4]$. However, this doesn't explain why the condition $\#_I[e'] \geq \#_I[e]$ is needed. In [5] we present an example showing that after removing this condition the tableau system is no longer complete.

Let K denote the number of reachable markings of Σ, and let B denote the maximum number of tokens that the reachable markings of Σ put in all the places of Σ. We have the following result:

Theorem 2. *Let $\mathcal{T}$ be a tableau of Σ constructed according to a total adequate order $\prec$.*

- *$\mathcal{T}$ is successful if and only if Σ has an illegal ω-trace.*
- *$\mathcal{T}$ contains at most $K^2 \cdot B$ non-terminal events.*
- *If the transitions of I are pairwise non-concurrent, then $\mathcal{T}$ contains at most K^2 non-terminal events.*

5 A Tableau System for the Illegal Livelock Problem

The tableau system for the illegal livelock problem is a bit more involved that that of the illegal ω-trace problem. In a first step we compute a set $CP = \{M_1, \ldots, M_n\}$ of reachable markings of Σ, called the set of *checkpoints*. This set has the following property: if Σ has an illegal livelock, then it also has an illegal livelock $M_0 \xrightarrow{t_1 t_2 \ldots t_i} M \xrightarrow{t_{i+1} t_{i+2} \ldots}$ such that $t_i \in L$ and M is a checkpoint. For the computation of CP we use the unfolding technique of [4] or [3]; the procedure is described in Sect. 5.1.

The tableau system solves the problem whether some checkpoint enables an infinite sequence of invisible actions. Clearly, Σ has an illegal livelock if and only if this is indeed the case. For this, we consider the net N_{inv} obtained from N by removing all the visible transitions together with their adjacent arcs. We construct unfoldings for the net systems $(N_{inv}, M_1), \ldots, (N_{inv}, M_n)$, and check

[4] We can also take the order of [3], which for this example yields the same results.

on them if the systems exhibit some infinite behavior. The tableau system is described in Sect. 5.2.

5.1 Computing the Set of Checkpoints

We construct the complete prefix of the unfolding of Σ as defined in [4] or [3]. In the terminology of this paper, the complete prefix corresponds to a tableau in which an event e is a terminal if there is an event e' such that $Mark([e']) = Mark([e])$, and $[e'] \prec [e]$.

Definition 4. *A marking M belongs to the set CP of* checkpoints *of Σ if $M = Mark([e])$ for some non-terminal event e of the complete prefix of Σ labelled by a transition of L.*

Let us compute CP for our example. The complete prefix of Σ coincides with the tableau for the illegal ω-trace problem. The events labelled by t_2, the only transition of L, are 2 and 11. The corresponding markings are $Mark([2]) = \{p_2, p_4\}$ and $Mark([11]) = \{p_4, p_7\}$. So $CP = \{ \{p_2, p_4\}, \{p_4, p_7\} \}$.

5.2 The Tableau System

Let $\{M_1, \ldots, M_n\}$ be the set of checkpoints obtained in the first phase. We will use $\Sigma_1, \ldots, \Sigma_n$ to denote the net systems $(N_{inv}, M_1), \ldots, (N_{inv}, M_n)$.

Definition 5. *Let $BP_1, \ldots, BP_n$ be branching processes of $\Sigma_1, \ldots, \Sigma_n$, respectively. An event e of BP_i is a* repeat *(with respect to $\prec$) if there is an index $j \leq i$ and an event e' in BP_j, called the* companion *of e, such that $Mark([e']) = Mark([e])$, and either*

(I) $j < i$, *or*
(II) $i = j$ *and* $e' < e$, *or*
(III) $i = j$, $\neg(e' < e)$, $[e'] \prec [e]$, *and* $|[e']| \geq |[e]|$.

A repeat e of BP_i is a terminal *if BP_i contains no repeat $e' < e$. Repeats, and in particular terminals, are of type I, II, or III, according to the condition they satisfy. A repeat e with companion e' is* successful *if it is of type II, and* unsuccessful *otherwise.*

A tableau *is a tuple $BP_1, \ldots, BP_n$ of branching processes of $\Sigma_1, \ldots, \Sigma_n$ such that for every $1 \leq i \leq n$ and for every possible extension e of BP_i at least one of the immediate causal predecessors of e is a terminal. Each BP_i is called a* tableau component. *A tableau is* successful *if at least one of its terminals is successful.*

Observe that an event of BP_i can be a repeat because of an event that belongs to *another* branching process BP_j. The definition of repeat depends on the order of the checkpoints, but the tableau system defined above is sound and complete for any fixed order. Because the definition of the tableau component

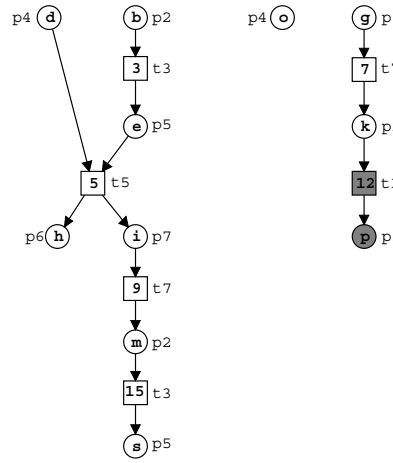


Fig. 3. The tableau system for the illegal livelock problem

BP_i depends only on the components with a smaller index, we can create the tableau components in increasing i order. Tableau components are constructed as for the illegal ω-trace problem, using the new definition of terminal.

The tableau for our example is shown in Fig. 3. The names of places and transitions have been chosen to match "pieces" of the unfolding in Fig. 2. The first tableau component contains no terminals; the construction terminates because no event labelled by an invisible transition can be added. In the second component, event 12 is a terminal with event 3 in the first component as companion. The intuition is that we don't need to unfold beyond 12 in the second component, because what we construct can be found after 3 in the first component.

Similarly to the case of the illegal ω-trace problem, a terminal of type II corresponds to a cycle in the reachability graph. Since the transitions of N_{inv} are all invisible, such a cycle always originates an illegal livelock, and so terminals of type II are always successful. For terminals of type III, the intuition is that $\uparrow[e]$ need not be constructed, because it is isomorphic to $\uparrow[e']$. The condition $|[e']| \geq |[e]|$ is required for completeness (see [5]). We have the following result:

Theorem 3. *Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be a tableau of $\Sigma_1, \ldots, \Sigma_n$ constructed according to a total adequate order $\prec$.*

- *$\mathcal{T}_1, \ldots, \mathcal{T}_n$ is successful if and only if Σ contains an illegal livelock.*
- *$\mathcal{T}_1, \ldots, \mathcal{T}_n$ contain together at most $K^2 \cdot B$ non-terminal events.*

5.3 A Tableau System for the 1-Safe Case

If Σ is 1-safe then we can modify the tableau system to obtain a bound of K^2 non-terminal events. We modify the definition of the repeats of type II and III:

(II') $i = j$ and $\neg(e' \# e)$, or

(III') $i = j$, $e' \# e$, $[e'] \prec [e]$, and $|[e']| \geq |[e]|$.

Theorem 4. *Let Σ be 1-safe. Let $\mathcal{T}_1, \ldots, \mathcal{T}_n$ be a tableau of $\Sigma_1, \ldots, \Sigma_n$ constructed according to a total adequate order $\prec$, and to the new definition of repeats of type II and III.*

- *$\mathcal{T}_1, \ldots, \mathcal{T}_n$ is successful if and only if Σ contains an illegal livelock.*
- *$\mathcal{T}_1, \ldots, \mathcal{T}_n$ contain together at most K^2 non-terminal events.*

6 A Tableau System for LTL Model Checking

Putting the tableau systems of Sections 4 and 5 together, we obtain a tableau system for the model checking problem of $tLTL'$. For the sake of clarity we have considered the illegal ω-trace problem and the illegal livelock problem separately. However, when implementing the tableau systems there is no reason to do so. Since all the branching processes we need to construct are "embedded" in the unfolding of $\Sigma_{\neg\varphi}$, it suffices in fact to construct *one single branching process*, namely the union of all the processes needed to solve both problems.

Clearly, this prefix contains $\mathcal{O}(K^2 \cdot B)$ non-terminal events. If the system is presented as a 1-safe Petri net, then the prefix contains $\mathcal{O}(K^2)$ non-terminal events because the following two conditions hold: (i) None of the reachable markings of the synchronization $\Sigma_{\neg\phi}$ enable two I-transitions concurrently. (ii) If the system is a 1-safe Petri net, then the synchronization $\Sigma_{\neg\phi}$ is also 1-safe.

7 Conclusions

We have presented a new unfolding technique for checking LTL-properties. We first make use of the automata-theoretic approach to model checking: a combined system is constructed as the product of the system itself and of an automaton for the negation of the property to be checked. The model checking problem reduces to the illegal ω-trace problem and to the illegal livelock problem for the combined system. Both problems are solved by constructing certain prefixes of the net unfolding of the combined system. In fact, it suffices to construct the union of these prefixes.

The prefixes can be seen as tableau systems for the illegal ω-trace and the illegal livelock problem. We have proved soundness and completeness of these tableau systems, and we have given an upper bound on the size of the tableau. For systems presented as 1-safe Petri nets or products of automata, tableaux contain at most size $\mathcal{O}(K^2)$ (non-terminal) events, where K is the number of reachable states of the system. An interesting open problem is the existence of a better tableau system such that tableaux contain at most $\mathcal{O}(K)$ events. We conjecture that it doesn't exist.

The main advantage of our approach is its simplicity. Wallner's approach proceeds in two steps: construction of a complete prefix, and then construction of a graph. The definition of a graph is non-trivial, and the graph itself can be exponential in the size of the complete prefix. Our approach makes the construction of the graph unnecessary. The price to pay is a larger prefix.

References

1. Bibliography on the net unfolding method. Available on the Internet at `http://wwwbrauer.in.tum.de/gruppen/theorie/pom/pom.shtml`.
2. J. Esparza. Model checking using net unfoldings. *Science of Computer Programming*, 23:151–195, 1994.
3. J. Esparza and S. Römer. An unfolding algorithm for synchronous products of transition systems. In *Proceedings of the 10th International Conference on Concurrency Theory (Concur'99)*, pages 2–20, 1999. LNCS 1055.
4. J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proceedings of 2nd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, pages 87–106, 1996. LNCS 1055.
5. Javier Esparza and Keijo Heljanko. A new unfolding approach to LTL model checking. Research Report A60, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, April 2000. Available at `http://www.tcs.hut.fi/pub/reports/A60.ps.gz`.
6. R. Gerth, D. Peled, M. Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *Proceedings of 15th Workshop Protocol Specification, Testing, and Verification*, pages 3–18, 1995.
7. B. Graves. Computing reachability properties hidden in finite net unfoldings. In *Proceedings of 17th Foundations of Software Technology and Theoretical Computer Science Conference*, pages 327–341, 1997. LNCS 1346.
8. K. Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets. In *Proceedings of 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, pages 240–254, 1999. LNCS 1579.
9. R. Kaivola. Using compositional preorders in the verification of sliding window protocol. In *Proceeding of 9th International Conference on Computer Aided Verification (CAV'97)*, pages 48–59, 1997. LNCS 1254.
10. R. Langerak and E. Brinksma. A complete finite prefix for process algebra. In *Proceeding of 11th International Conference on Computer Aided Verification (CAV'99)*, pages 184–195, 1999. LNCS 1663.
11. K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.
12. S. Melzer and S. Römer. Deadlock checking using net unfoldings. In *Proceedings of 9th International Conference on Computer-Aided Verification (CAV '97)*, pages 352–363, 1997. LNCS 1254.
13. P. Niebert. Personal communication, 1999.
14. C. Stirling and David Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.
15. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, pages 238–265, 1996. LNCS 1043.
16. F. Wallner. Model checking techniques using net unfoldings. PhD thesis, Technische Universität München, Germany, forthcoming.
17. F. Wallner. Model checking LTL using net unfoldings. In *Proceeding of 10th International Conference on Computer Aided Verification (CAV'98)*, pages 207–218, 1998. LNCS 1427.
18. P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1,2):72–93, 1983.

Reasoning about Message Passing in Finite State Environments *

B. Meenakshi and R. Ramanujam

The Institute of Mathematical Sciences
C.I.T. Campus, Chennai 600 113
India.
{bmeena,jam}@imsc.ernet.in

Abstract. We consider the problem of reasoning about message based systems in finite state environments. Two notions of finite state environments are discussed: bounded buffers and implicit buffers. The former notion is standard, whereby the sender gets blocked when the buffer is full. In the latter, the sender proceeds as if the buffer were unbounded, but the system has bounded memory and hence "forgets" some of the messages. The computations of such systems are given as communication diagrams. We present a linear time temporal logic which is interpreted on n-agent diagrams. The formulas of the logic specify local properties using standard temporal modalities and a basic communication modality. The satisfiability and model checking problems for the logic are shown to be decidable for both buffered products and implicit products. An example of system specification in the logic is discussed.

1 Motivation

In distributed systems, the computations of autonomous agents proceed asynchronously and communication is by message passing. When the medium of communication buffers messages, the sender is assumed to put messages into the medium and proceed in its computation whereas a receiver may need to wait for message delivery. This notion that the sender is not blocked is often referred to as the *unbounded buffer abstraction* ([LL90]), and is widely used in the design of agents in distributed systems. In addition, assumptions are made about whether message delivery is guaranteed or not, about whether messages are delivered in the order in which they were sent, and so on. Most distributed algorithms work on variants of such models ([Lyn96]).

While the assumption that buffers are unbounded abstracts away messy details and hence is welcome, implementations need to work with bounded buffers. But then, when such a buffer is full, the sender must wait till one of the elements has been removed. We need to modify the original design, and since each change may cascade a number of changes in the pattern of communication

* We thank Kamal Lodaya and the reviewers for helpful discussions, and Laura Semini for bringing to our attention [MS99] on which Section 5 is based.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 487–498, 2000.
© Springer-Verlag Berlin Heidelberg 2000

between agents, we may end up with deadlock or other disastrous possibilities, necessitating verification again. Therefore, we would like to have a methodology whereby systems are developed without reference to buffers, and system properties are stated without mentioning buffer status, but buffers figure only in the analysis. In some sense, buffer behaviour should be abstracted as environment behaviour, where the environment is (hopefully) finite state.

When both the agents in the system and the environment are finite state, *automata-theoretic techniques* can be employed in the verification of *temporal* properties [MP95]. In this paper, we follow such a framework in the context of message passing in finite state environments. We consider systems of finite state automata with communication constraints between them. The computations of these systems are given as communication diagrams, a slight generalization of Lamport diagrams [La78] and message sequence charts (MSCs). We study two notions of finite state environments: product of automata defined by bounded buffers and that defined by implicit buffers. The former notion is standard, whereby the sender gets blocked when the buffer is full. In the latter, the sender proceeds as if the buffer were unbounded, but the system has bounded memory and hence "forgets" some of the messages, or the order in which messages were sent. In our automaton model, communications are not observable system transitions, but only define a coupling relation that constrains the possible runs of the system; this is a departure from the way such systems are typically modelled in process algebras or net theory.

The diagrams suggest a natural linear time temporal logic in which the formulas specify local properties using standard temporal modalities and a basic communication modality. The satisfiability and model checking problems for the logic are shown to be decidable for both buffered products and implicit products. We illustrate how this logic can be employed in reasoning about message based systems using a conference coordination example suggested by [MS99], [CNT98]. This system consists of authors who submit papers, reviewers who examine the papers and a moderator who communicates the results.

A distinct feature of the proposed logic is that it uses local assertions as in [Ra96]. In the context of bounded buffer products, this means that the logic cannot force buffers of specific size. Most of the locally based logics in the literature study systems where communication is only by synchronous means, or, as is the case with [LRT92], study subclasses of event structures, where the model checking problem is not addressed. The work of [HNW99] does consider the model checking problem using finite unfoldings of a class of nets, but since they do not study linearizations of the associated partial order, these difficulties related to buffers do not arise there.

[AY99] also present results about model checking message passing systems where the computations are MSCs. The difference is that our approach is dual: as [AY99] put it neatly, the communicating state machine model (that we study) is a parallel composition of sequential machines, while the MSC model (that they discuss) is a sequential composition of concurrent executions.

2 Communicating Automata

Throughout the paper, we fix $n > 0$, and study distributed systems of n agents. We will follow the linguistic convention that the term 'system' will refer to composite distributed systems, and 'agent' to component processes in systems. In any system, the agents are referred to by the indices i, $1 \leq i \leq n$. We will use the notation $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

A *distributed alphabet* for such systems is an n-tuple $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_n)$, where for each $i \in [n]$, Σ_i is a finite nonempty alphabet of *i-actions* and for all $i \neq j$, $\Sigma_i \cap \Sigma_j = \emptyset$. The set of *system actions* is the set $\Sigma' = \{\lambda\} \cup \Sigma$, where $\Sigma = \bigcup_i \Sigma_i$. λ is referred to as the communication action. We will use a, b, c etc to refer to elements of Σ and τ, τ' etc to refer to those of Σ'.

Definition 2.1 *A* **System of n communicating automata (SCA)** *on $\widetilde{\Sigma}$ is a tuple $S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$, where for $j \in [n]$, Q_j is a finite set of states, $G_j \subseteq Q_j$, $Init \subseteq (Q_1 \times \ldots \times Q_n)$ and for $i \neq j$, $Q_i \cap Q_j = \emptyset$. Let $Q = \bigcup_i Q_i$. $\rightarrow \subseteq (Q \times \Sigma' \times Q)$ such that if $q \xrightarrow{\tau} q'$ then either there exists i such that $\{q, q'\} \subseteq Q_i$ and $\tau \in \Sigma_i$, or there exist $i \neq j$ such that $q \in Q_i, q' \in Q_j$ and $\tau = \lambda$.*

$Init$ contains the (global) initial states of the system and G_j are the (local) good states of agent j. We will use the notation ${}^{\bullet}q \stackrel{\text{def}}{=} \{q' \mid q' \xrightarrow{\lambda} q\}$ and $q^{\bullet} \stackrel{\text{def}}{=} \{q' \mid q \xrightarrow{\lambda} q'\}$. The set $\widetilde{Q} \stackrel{\text{def}}{=} (Q_1 \times \ldots \times Q_n)$, refers to global states, and when $u = (q_1, \ldots, q_n) \in \widetilde{Q}$, $u[i]$ refers to q_i.

Note that $\rightarrow$ above is *not* a global transition relation, it consists of local transition relations, one for each automaton, and communication constraints of the form $q \xrightarrow{\lambda} q'$, where q and q' are states of different automata. The latter define a coupling relation rather than a transition. The interpretation of local transition relations is standard: when the ith automaton is in state q_1 and reads input $a \in \Sigma_i$, it can move to a state q_2 and be ready for the next input if $(q_1, a, q_2) \in \rightarrow$. The interpretation of the communication constraint is non-standard and depends only on automaton states, not on input. When $q \xrightarrow{\lambda} q'$, where $q \in Q_i$ and $q' \in Q_j$, it constrains the system behaviour as follows: whenever automaton i is in state q, it puts out a message whose content is q and intended recipient is j; whenever automaton j needs to enter state q', it checks its environment to see if a message of the form q from i is available for it, and waits indefinitely otherwise. If a system S has no λ constraints at all, the automata in it proceed asynchronously and do not wait for each other.

However, computations where every receipt of a message is matched with a corresponding 'send' lead to non-regular behaviour. We therefore constrain the interpretation of λ constraints to reflect *finite environments*. The first one we consider is that of bounded buffers. When $q \xrightarrow{\lambda} q', q \in Q_i, q' \in Q_j$, when i needs to enter state q, it checks whether the bounded buffer constituting the environment

is full; if it is, then it waits till it is no longer full, then puts in a message q for j and proceeds. The product construction defined below is designed to reflect this intuition.

2.1 Bounded Buffers

To keep the presentation simple, we will focus only on 1-buffered runs. (In the next section, we will see that unit buffers suffice for the logic presented, justifying this decision.) The product definition below can be generalized for k-buffered runs, for any $k > 0$, but becomes technically more involved.

Definition 2.2 *Given an SCA* $S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$ *on* $\widetilde{\Sigma}$*, the* **1-product** *of the system is defined to be* $Pr_S = (X, \widetilde{I}, \widehat{G}, \Rightarrow)$*, where* $X = \widetilde{Q} \times \mathcal{B}$*,* $\widetilde{I} = (Init \times \{\emptyset\})$*,* $\widehat{G} = (G_1, \ldots, G_n)$ *and* $\mathcal{B} = \{B \subseteq ([n] \times Q) \mid$ *if* $(i, q) \in B$ *then* $q \notin Q_i$*, and for each* i, j*, there exists at most one* $(i, q) \in B$ *where* $q \in Q_j\}$*.* $\Rightarrow \subseteq (X \times \Sigma \times X)$ *is defined by:* $(q_1, \ldots, q_n, B) \overset{a}{\Rightarrow} (q'_1, \ldots, q'_n, B')$*,* $a \in \Sigma_i$*, iff*

1. $q_i \overset{a}{\rightarrow} q'_i$*, and for all* $j \neq i, q_j = q'_j$*.*
2. *if* $({}^{\bullet}q'_i \cap Q_j) = R \neq \emptyset$*, then there exists* $q \in R$ *such that* $(i, q) \in B$ *and* $B' = B - \{(i, q)\}$*.*
3. *if* $(q_i{}^{\bullet} \cap Q_j) \neq \emptyset$*, then* $\{(j, q) \in B \mid q \in Q_i\} = \emptyset$ *and* $B' = B \cup \{(j, q_i)\}$*.*

A state $q \in Q_i$ is said to be terminal if $\{q' \in Q_i \mid q \overset{a}{\rightarrow} q'$ for some $a \in \Sigma_i\} = \emptyset$. For $w \in \Sigma^\omega$, we use the notation $w\lceil i$ to denote the restriction of w to Σ_i.

Computations of S are defined by runs of Pr_S. We consider only infinite behaviours in this paper. Let $w = a_1 a_2 \ldots \in \Sigma^\omega$. An infinite run $\rho = x_0 x_1 \ldots$ on w is a sequence where for $k \geq 0$, $x_k \overset{a_{k+1}}{\Rightarrow} x_{k+1}$. We say that i terminates in ρ if there exists k such that $x_k[i]$ is terminal. ρ is said to be *good* if for all $i \in [n]$, $w\lceil i$ is infinite or i terminates in ρ. Let $Inf_i(\rho) \overset{\text{def}}{=} \{q \in Q_i \mid$ for infinitely many $k \geq 0$, $x_k[i] = q\}$. The run ρ on w is said to be *accepting* iff ρ is good, $x_0 \in \widetilde{I}$, and for all i, $Inf_i(\rho) \cap G_i \neq \emptyset$. The *language accepted by* Pr_S, denoted $\mathcal{L}^1(S) \overset{\text{def}}{=} \{w \in \Sigma^\omega \mid S$ has an accepting run on $w\}$. Note that, if Pr_S has no infinite runs at all, then $\mathcal{L}^1(S) = \emptyset$. A consequence of the definition of good runs is that no agent gets stuck because of the buffer condition. Figure 1 gives a 2-agent system and its 1-buffered product.

Even though we refer to these products as 1-buffered, there are really $n(n-1)$ unit buffers in the system, one for each pair $(i, j), i \neq j$, each containing at most one message. Emptiness checking for an SCA can be done in time linear in the size of the constructed product. We have:

Lemma 2.3 *Given an SCA* S *of* n *automata, checking whether* $\mathcal{L}^1(S) \overset{?}{=} \emptyset$ *can be done in time* $k^{O(n)}$*, where* k *is the maximum of* $\{|Q_i| \mid i \in [n]\}$*.*

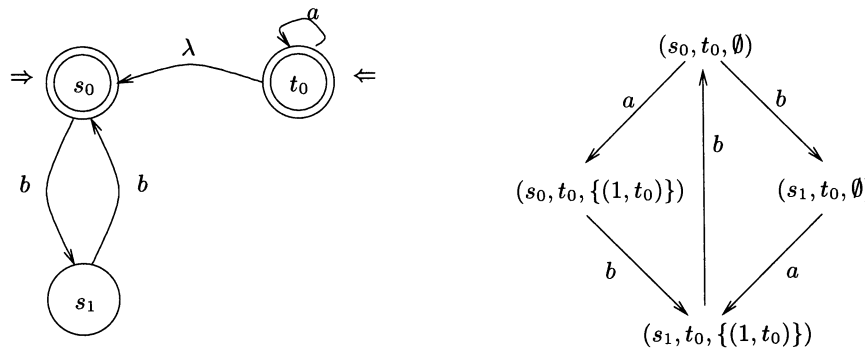


Fig. 1. 1-buffered product

2.2 Implicit Buffers

We now study an alternative to bounded buffers, based on a more abstract notion of finite state environment. For this, recall the notion of messages in SCAs: an edge $q \xrightarrow{\lambda} q'$, with $q \in Q_i$ and $q' \in Q_j$, represents a constraint on runs; q' cannot be visited in the run until q has been visited prior to it. The difficulty comes when q is visited several times; if each visit is recorded as a message in the buffer, unless i is made to wait, we require unbounded memory. On the other hand, if i is in a cycle, should we consider each visit as a new message ? Or, if i cycles between q_0 and q_1, is it important for the receiver to know whether the cycle was entered at q_0 or at q_1 ? These considerations lead us to the notion of implicit products of automata in systems below.

Definition 2.4 *Given an SCA* $S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$ *on* $\widetilde{\Sigma}$, *the* **implicit buffered product** *of the system is defined to be the tuple* $Pr_S^{imp} = (X, \widetilde{I}, \widehat{G}, \Rightarrow)$, *where* $X = \widetilde{Q} \times \mathcal{B}$, $\widetilde{I} = (Init \times \{\emptyset\})$, $\widehat{G} = (G_1, \ldots, G_n)$ *and* $\mathcal{B} = \{B \subseteq ([n] \times Q) \mid$ *if* $(i, q) \in B$ *then* $q \notin Q_i\}$. $\Rightarrow \subseteq (X \times \Sigma \times X)$ *is defined by:* $(q_1, \ldots, q_n, B) \overset{a}{\Rightarrow} (q'_1, \ldots, q'_n, B')$, $a \in \Sigma_i$, *iff*

1. $q_i \xrightarrow{a} q'_i$, *and for all* $j \neq i, q_j = q'_j$.
2. *if* $({}^\bullet q'_i \cap Q_j) = R \neq \emptyset$, *then there exists* $q \in R$ *such that* $(i, q) \in B$ *and* $B' = B - \{(i, q)\}$.
3. *if* $(q_i{}^\bullet \cap Q_j) \neq \emptyset$, *then* $B' = B \cup \{(j, q_i)\}$.

The language accepted by the system, denoted $\mathcal{L}_{imp}(S)$, is defined exactly as before. Clearly, we have a notion here that's very different from the previous one. In fact it is easy to construct an SCA S such that $\mathcal{L}_{imp}(S)$ is infinite, but $\mathcal{L}^1(S)$ is empty, with the system deadlocked. In implicit buffered products, the sender is never blocked. However, every agent i can offer a subset of Q_i for each $j \neq i$ as messages in the buffer, and hence the complexity of checking for emptiness is higher.

Lemma 2.5 *Given an SCA S of n automata, checking whether $\mathcal{L}_{imp}(S) \stackrel{?}{=} \emptyset$ can be done in time $2^{O(kn^2)}$, where k is the maximum of $\{|Q_i| \mid i \in [n]\}$.*

3 Lamport Diagrams and a Temporal Logic

The computations of systems of communicating automata can be seen as Lamport diagrams, which depict local runs and message dependence between them.

Definition 3.1 *A* **Lamport diagram** *is a tuple $D = (E_1, \ldots, E_n, \leq)$, where E_i is the set of* event occurrences *of agent i, $\leq$ is a partial order on $E \stackrel{\text{def}}{=} \bigcup_i E_i$ called the* causality relation *such that: for all $i \in \{1, \ldots, n\}$, E_i is totally ordered by $\leq$, and for all $e \in E$, $\downarrow e \stackrel{\text{def}}{=} \{e' \mid e' \leq e\}$ is finite.*

Since for all $e \in E$, $\downarrow e$ is finite, $\leq$ must be discrete. Hence there exists $\lessdot \subset \leq$, the *immediate causality relation*, which generates the causality relation; that is: for all e, e', e'', if $e \lessdot e'$ and $e \leq e'' \leq e'$ then $e'' \in \{e, e'\}$. We have: $\leq = (\lessdot)^*$. When $e \in E_i$, $e' \in E_j$, $i \neq j$ and $e \lessdot e'$, we interpret e as the sending of a message by agent i and e' its corresponding receipt by j.

Let $e \in E_i$. We can think of $\downarrow e$ as the local state of agent i when the event e has just occurred. This state contains the information that i has up till that instant in the computation, which contains it own local history and that of others according to the latest communication from them. The empty set corresponds to the initial state, where no i-event has occurred, and is denoted by ϵ_i. Let the set of all local states of agent i be denoted $LC_i \stackrel{\text{def}}{=} \{\epsilon_i\} \cup \{\downarrow e \mid e \in E_i\}$ and let $LC \stackrel{\text{def}}{=} \bigcup_i LC_i$. We use d, d' etc to denote local states. We can extend the $\lessdot$ relation to local states as follows: let $d_1 \in LC_j$ and $d_2 \in LC_i$; we say $d_1 \lessdot d_2$ iff $d_1 \subseteq d_2$ and for all $d \in LC_j$, if $d \subseteq d_2$, then $d \subseteq d_1$ as well; that is, d_1 is the last j-local state seen by i at d_2.

Given a Lamport diagram $D = (E_1, \ldots, E_n, \leq)$, a *sequentialization* of D is any sequence $\sigma = e_0 e_1 \ldots$ such that $E = \{e_0, e_1, \ldots\}$ and for all $k \geq 0$, $\downarrow e_k \subseteq \{e_0, \ldots, e_k\}$; that is, σ is a linear order that respects $\leq$. Let $\sigma = e_0 e_1 \ldots$ be a sequentialization of D. We say σ is *k-bounded* iff the following property holds: suppose $\{h_1, \ldots, h_k, h_{k+1}\} \subseteq E_i$ such that $h_1 \leq \ldots \leq h_{k+1}$, for all $l : 1 \leq l \leq k$, g_l is the j-maximal event in $\downarrow h$, where h is any event such that $h_l \leq h < h_{l+1}$, and g_{k+1} is the j-maximal event in $\downarrow h_{k+1}$; then g_{k+1} occurs later than h_1 in σ. It is easy to see that, when $k = 1$, this is basically the same notion as that of 1-buffered runs studied in the previous section.

3.1 From Runs to Lamport Diagrams

Consider an SCA S; suppose that Pr_S has an infinite run $\rho = x_0 x_1 \ldots$, on $w = a_1 a_2 \ldots \in \Sigma^\omega$, i.e., for $k \geq 0$, $x_k \stackrel{a_{k+1}}{\Rightarrow} x_{k+1}$. ρ induces a clock function $\chi : ([n] \times [n] \times \mathbf{N}) \to \mathbf{N}$ which records, for each pair of agents i, j and each

instance k, the *latest* instant at which i last heard from the agent j at k. We define $\chi(i,j,k)$ by induction on k. Set $\chi(i,j,0) = 0$ for all i,j. Suppose $\chi(i,j,k)$ is inductively defined and ${}^{\bullet}\rho(k+1)[i] \cap Q_j \neq \emptyset$; then, between $\chi(i,j,k)$ and k, there could be many instances l when j could have sent a 'message' to i ($\rho[l](j)^{\bullet} \cap Q_i \neq \emptyset$). However, since ρ is 1-buffered, l is unique and $\chi(i,j,k+1)$ is set to l.

The Lamport diagram D_ρ associated with ρ is defined as follows. Every transition $x_k \overset{a_{k+1}}{\Rightarrow} x_{k+1}$ of ρ is recorded as an event $(k,k+1)$ of E_i in D_ρ, where $a_{k+1} \in \Sigma_i$. The local causal order $\leq_i$ of agent i consists of all pairs of the form $((k,k+1),(l,l+1))$ such that $k \leq l$. For communication edges, define $(m-1,m) \lessdot_c (k,k+1)$ iff $(m-1,m) \in E_j$, $(k,k+1) \in E_i$, $i \neq j$ and $\chi(i,j,k) < \chi(i,j,k+1) = m$. Now we define $\leq = ((\bigcup_i \leq_i) \cup \lessdot_c)^*$ and it is easy to see that D_ρ is a Lamport diagram. Thus, to each infinite run ρ of Pr_S, we can associate a Lamport diagram D_ρ.

3.2 A Temporal Logic

We now proceed to define the temporal logic, which we call **m-LTL** in which we can reason about local assertions on Lamport diagrams. A crucial aspect of the logic is the asymmetry in the communication modality: *the receiver of a message gets information about the sender's past, whereas the sender cannot access the receiver's future.* This is because, in distributed systems, when we cannot make any assumptions about relative speeds of processes, the sender cannot, in general, infer any information about the status of the receiver at the time of message receipt. On the other hand, local information (like, "I have sent a message") can always be maintained using local propositions by the sender.

Fix countable sets of *propositional letters* $(P_1, P_2, \ldots, P_n)$, where P_i consists of the atomic local properties of agent i. Let $P \overset{\text{def}}{=} \bigcup_i P_i$.

Let $i \in [n]$. The syntax of i-local formulas is given below:

$$\Phi_i ::= p \in P_i \mid \neg\, \alpha \mid \alpha_1 \vee \alpha_2 \mid \bigcirc\, \alpha \mid \alpha_1 \,\mathbf{U}\, \alpha_2 \mid \oslash_j\, \alpha,\ j \neq i,\ \alpha \in \Phi_j$$

Global formulas are obtained by boolean combination of local formulas:

$$\Psi ::= \alpha @ i,\ \alpha \in \Phi_i \mid \neg\, \psi \mid \psi_1 \vee \psi_2$$

The propositional connectives $(\wedge, \supset, \equiv)$ and derived temporal modalities $(\Diamond, \Box)$ are defined as usual. The formulas are interpreted on Lamport diagrams. For technical convenience, we consider only infinite behaviours. Formally, models are pairs of the form $M = (D, V)$, where $D = (E_1, \ldots, E_n, \leq)$ is a Lamport diagram such that $E = \bigcup_i E_i$ is a countably infinite set and $V : LC \to 2^P$ is the valuation map such that for $d \in LC_i$, $V(d) \subseteq P_i$.

Let $\alpha \in \Phi_i$ and $d \in LC_i$. The notion that α holds in the local state d of agent i in model M is denoted $M, d \models_i \alpha$, and is defined inductively as usual:

- $M, d \models_i p$ iff $p \in V(d)$.
- $M, d \models_i \neg\alpha$ iff $M, d \not\models_i \alpha$.
- $M, d \models_i \alpha \vee \beta$ iff $M, d \models_i \alpha$ or $M, d \models_i \beta$.
- $M, d \models_i \bigcirc\alpha$ iff there exists $d' \in LC_i$ such that $d \lessdot d'$ and $M, d' \models_i \alpha$.
- $M, d \models_i \alpha \mathbf{U} \beta$ iff $\exists d' \in LC_i$: $d \subseteq d', M, d' \models_i \beta$ and $\forall d'' \in LC_i : d \subseteq d'' \subset d' : M, d'' \models_i \alpha$.
- $M, d \models_i \oslash_j \alpha$ iff there exists $d' \in LC_j$ such that $d' \lessdot d$ and $M, d' \models_j \alpha$.

The new modality $\oslash_j \alpha$, asserted by i, says that α held in the last j-local state visible to i. For global formulas, the corresponding notion is defined only at initial states, in terms of the notion defined above for local formulas.

- $M \models \alpha@i$ iff $M, \epsilon_i \models_i \alpha$; $M \models \neg\psi$ iff $M \not\models \psi$;
  $M \models \psi_1 \vee \psi_2$ iff $M \models \psi_1$ or $M \models \psi_2$.

We say that ψ is *satisfiable* iff there exists a model M such that $M \models \psi$. We say that ψ is *valid* if for every model M, we have $M \models \psi$.

A typical specification in the logic has the form: $(\Box(p \wedge \oslash_2 \neg \text{`}OK\text{'} \supset \bigcirc(q \wedge \oslash_2 \text{`}OK\text{'})))@1$, which asserts that agent 1 can make a transition from a state satisfying p into a state in which q holds only after hearing an 'OK' from agent 2, and must block otherwise.

3.3 Model Checking

Every (1-buffered) infinite run of an SCA gives rise to a Lamport diagram, and formulas of m-LTL are interpreted over such diagrams. We are thus in a position to formulate the problem we began with: can we automatically check that all runs of a given SCA satisfy a specification ψ in the logic m-LTL ?

To make this precise, we define an *interpreted system* to be a pair $\mathcal{S} = (S, Val)$, where $S = ((Q_1, G_1), \ldots, (Q_n, G_n), \rightarrow, Init)$ on $\widetilde{\Sigma}$, $Val : Q \rightarrow 2^P$ such that for all $q \in Q_i$, $Val(q) \subseteq P_i$. Now, every infinite run $\rho = x_0 x_1 \ldots$ of S defines a Lamport diagram D_ρ as we have seen above; we define the associated model $M_\rho = (D_\rho, V_\rho)$, where $V_\rho(\epsilon_i) \stackrel{\text{def}}{=} Val(x_0[i])$, and for all $e = (k, k+1)$ in E_i, $V_\rho(\downarrow e) \stackrel{\text{def}}{=} Val(x_{k+1}[i])$. It can be easily checked that V_ρ is indeed a legal valuation. We say that an interpreted system $\mathcal{S} = (S, Val)$ satisfies a formula ψ of m-LTL iff for every accepting run ρ of S, the associated model (D_ρ, V_ρ) induced by Val and S satisfies ψ. We denote this by $\mathcal{S} \models \psi$.

Theorem 3.2 *Let ψ be a m-LTL formula of length m. Satisfiability of ψ over n-agent Lamport diagrams can be checked in time $2^{O(mn)}$. Let $\mathcal{S}$ be an interpreted SCA with k being the maximum of $\{|Q_i| \mid i \in [n]\}$. Then the question $\mathcal{S} \models^? \psi$ can be answered in time $k^{O(n)}.2^{O(mn)}$.*

The theorem is proved by associating a system S_ψ over the distributed alphabet $(2^{P_1}, \ldots, 2^{P_n})$ with every formula ψ such that $\mathcal{L}^1(S_\psi)$ in some way corresponds exactly to the class of models of ψ. This correspondence is made precise

as follows: let $M \models \psi$, where $M = (D, V)$ and let $\sigma = e_1 e_2 \ldots$ be any 1-bounded sequentialization of D. Let $Prop_\sigma = V(\downarrow e_1)V(\downarrow e_2)\ldots$. Now, define $Lang(\psi)$ to be the set of all such $Prop_\sigma$; it is a subset of $(2^P)^\omega$. Moreover, M defines an n-tuple $< V(\epsilon_1), \ldots, V(\epsilon_n) >$. Let $\widehat{V}(\psi)$ denote the set of all such tuples. Then what we do is to associate an interpreted system (S_ψ, Val_ψ) with every formula ψ such that $\mathcal{L}^1(S_\psi) = Lang(\psi)$ and $Val_\psi(Init) = \widehat{V}(\psi)$, where $Init$ is the set of global initial states of S and Val is applied pointwise on it.

Now, checking decidability of a formula amounts to checking whether the associated system accepts a nonempty language. For model checking, first note that languages accepted by 1-buffered products are closed under intersection. Hence, we can check emptiness of the language obtained by the intersection $\mathcal{L}^1(S) \cap \mathcal{L}^1(S_{\neg\psi})$ and answer accordingly whether the given interpreted system satisfies ψ or not. (Proof details are available from the authors.)

4 Implicit Buffers

The computations of implicit products cannot be presented as Lamport diagrams, since 'communication edges' can cross each other now, as they do in systems where messages are not guaranteed to be delivered in the order in which they were sent. Moreover, since repeated visits to 'send' states are possible without ever visiting corresponding 'receive' states, the modelled systems do not provide guaranteed message delivery either. We now generalize Lamport diagrams to include these possibilities formally and reinterpret our logic on them.

Definition 4.1 *A* **communication diagram** *is a tuple* $C = ((E_1, \leq_1), \ldots, (E_n, \leq_n), <_c)$*, where for all* $i \in [n]$*,* $\leq_i$ *is a total order on* E_i *and* $<_c \subset (E \times E)$ *where* $E = \bigcup_i E_i$*. For* $i \neq j$*,* $E_i \cap E_j = \emptyset$ *and* $<_c$ *satisfies the following conditions:*

1. $\leq = (\bigcup_i \leq_i \cup <_c)^*$ *is acyclic.*
2. *If* $e <_c e'$ *and* $e \in E_i$*, then* $e' \notin E_i$*, and*
3. *For all* $e \in E_i$*, for each* $j \neq i$*,* e *has at most one* $<_c$ *successor and at most one* $<_c$ *predecessor in* E_j*.*

Above, we refer to $(E, \leq)$ as the poset generated by C.

Given any Lamport diagram $D = (E_1, \ldots, E_n, \leq)$, the structure defined by $C_D = ((E_1, \leq_1), \ldots, (E_n, \leq_n), <_c)$ is a communication diagram, where $\leq_i = \leq \cap (E_i \times E_i)$ and $<_c = \lessdot - \bigcup_i \leq_i$. On the other hand, given any communication diagram C, we can associate a structure $D_C = (E_1, \ldots, E_n, \leq)$ with it, where $\leq$ is the partial order generated by C. However, there is an important difference between these classes of diagrams, as the following proposition shows.

Proposition 4.2 *Let* D*,* C *be as above:* D_{C_D} *is isomorphic to* D*.* C_{D_C} *is isomorphic to* C *only if* C *satisfies the following condition: for all* $e_1, e_2 \in E_i$*,* $f_1, f_2 \in E_j$*,* $i \neq j$*, if* $e_1 \leq_i e_2$*,* $e_1 <_c f_1$ *and* $e_2 <_c f_2$*, then* $f_1 \leq_j f_2$*.*

Thus communication diagrams generalize Lamport diagrams. We can now interpret m-LTL on these frames. The notion $\downarrow e$ now refers to the generated relation $\leq$. The only changes are in the semantics of the communication modality.

- Let $d \in LC_i$ and $j \neq i$.

$$M, d \models_i \oslash_j \alpha \text{ iff } \exists e \in E_i : d = \downarrow e, \exists e' \in E_j : e' <_c e \text{ and } M, \downarrow e' \models_j \alpha$$

- An implicit model is a pair $M = (C, V)$, where C is a communication diagram such that E is countable, and $V : LC \to 2^P$ is the valuation map such that for $d \in LC_i$, $V(d) \subseteq P_i$.

We now speak of implicit models for a formula, and of a formula being implicitly satisfiable. Note that the $\oslash_j$ modality now refers to the presence of an explicit edge. e' may be the j-maximal event occurrence in $\downarrow e$ without $e' \lessdot e$; in the earlier semantics such an e' could witness the truth of $\oslash_j \alpha$ at $\downarrow e$, but this is not possible with the semantics presented here.

Given a formula ψ_0, we can construct a system S_{ψ_0} in a similar manner as before. We omit the proof and mention only the results.

Theorem 4.3 *Implicit satisfiability of a formula ψ in m-LTL over n-agent communication diagrams can be decided in time $2^{O(2^m n^2)}$, where $m = |\psi|$. Let $\mathcal{S}$ be an interpreted SCA with k being the maximum of $\{|Q_i| \mid i \in [n]\}$. Then the question $\mathcal{S} \models^?_{imp} \psi$ can be answered in time $k^{O(n)}.2^{O(2^m n^2)}$.*

5 A System Specification

In this section we consider a simplified version of a case study [CNT98], [MS99]. The system has five components — two authors (A_1 and A_2), a moderator M and two reviewers (R_1 and R_2). Both authors submit papers to M who passes them on to both reviewers who review each submission, and send the result to M. A paper is accepted only if both the reviewers choose to accept it. M communicates results to authors and there is no direct communication between authors and reviewers. For simplification we assume that papers are considered one at a time.

The system is modelled as an SCA given in Figure 2. For simplification, we have just shown the automata corresponding to A_1, M, R_1. The λ-constraints across the system are shown by dotted-directed lines pointing to the sending/receiving states. The λ-constraints associated with the states p'_i and s'_j (of M and R_1 respectively) are symmetric to those associated with the states p_i and s_j respectively and hence not mentioned.

A_1 submits at state q_0 (a λ constraint to state p_1 of M), and then waits for result: accept is state q_1 (λ edge from p_6 of M), reject is q_2, and a positive result for author is A_2 is recorded in q_3. M has two symmetric sub-components, one for A_1 and the other for A_2. When it gets a submission (state p_1), it sends it to R_1 and R_2 and waits for the four possible outcomes, one of which is accept

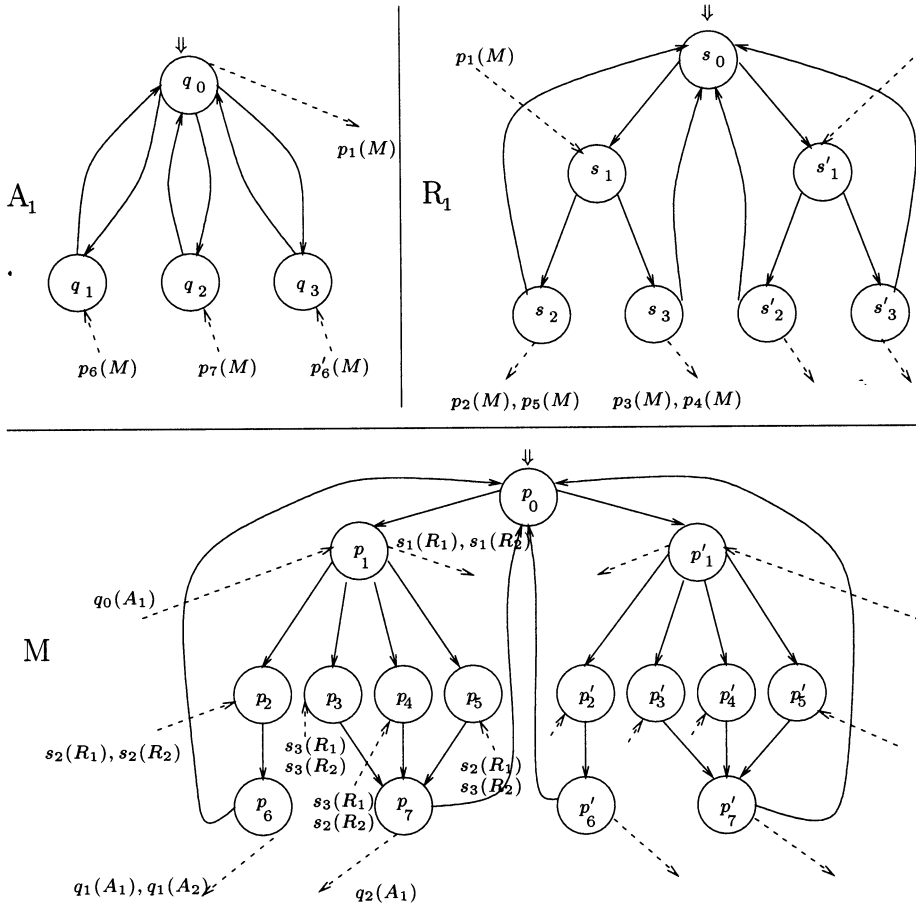


Fig. 2. Paper review system

(state p_2), the rest being reject (states p_3 through p_5); these are accordingly communicated in states p_6 and p_7. R_1 is simple: it gets a submission, makes a binary choice and reverts to waiting.

With each component a set of propositions is associated. The intended meaning of each proposition should be clear from the name. With A_1: $P_1 = \{result_1, pl_1 = \emptyset, pl_1 = 1, pl_1 = 2\}$. With M: $P_3 = \{wait, proc_1, proc_2\} \cup \{FB_i = x \mid x \in Dec, i = 1,2\} \cup \{R = x \mid x \in Dec\}$ where $Dec = \{acc, rej, nil\}$. With R_1: $P_4 = \{ready_1, review_1(i), decided_1(i), OK_1(i) \mid i = 1,2\}$. The set of propositions P_2 for author A_2 and P_5 for reviewer R_2 correspond to P_1 and P_4 respectively. pl_i indicates the 'publication list' of i. Note that each author keeps a record of the current list. FB_i stands for feedback from i.

The valuation map is suitably defined from the automaton description. The following are some examples of local properties expressed in m-LTL, which the 1-buffered product of the system in Figure 2 satisfies.

1. M-formula: $\Box(proc_1 \wedge \oslash_1(\neg result_1 \wedge pl_1 = \emptyset) \supset ((proc_1 \wedge \neg wait)\mathbf{U}(R = acc \vee R = rej))))$
2. M-formula: $\Box((proc_1 \wedge (R = acc)) \supset (\oslash_4(decided_1(1) \wedge OK_1(1)) \wedge \oslash_5(decided_2(1) \wedge OK_2(1))))$
3. A_1-formula: $\Box(pl_1 = 2 \supset \oslash_3(proc_2 \wedge R = acc))$
4. R_1-formula: $\Box(review_1(1) \supset \oslash_1(\neg result_1 \wedge pl_1 = \emptyset))$

Interestingly, R_1 can assert this despite having no direct contact with A_1: this is because it can assert $\oslash_3\oslash_1(\neg result_1 \wedge pl_1 = \emptyset)$ above, and by the semantics of the modality, the use of $\oslash_1$ suffices.

Remark: m-LTL is based on local assertions, and hence the sender of a message cannot refer to the state of the receiver on receiving the message. However we can use propositions $s^i_j \in P_i$, $i \neq j$, whereby i asserts that a message meant for j has been sent; this is a local property and does not refer to receiver states. The technical results can be easily extended, and we get additional convenience in specification. We can also study extensions of the logic with past tense and global modalities; in the presence of the 'send'-propositions above, global formulas can force buffers of specified size in buffered products.

References

AY99. Alur, R. and Yannakakis, M., "Model checking of message sequence charts", Springer *LNCS 1664*, 1999, 114-129.

CNT98. Ciancarini, P., Nierstrasz, O. and Tolksdorf, R., "A case study in co-ordination: conference management on the internet", Available at: http://malvasia.di.fct.unl.pt/activity/coordi na/working/case-studies.

HNW99. Huhn, M., Niebert, P. and Wallner, F., "Model checking logics for communicating sequential agents", Springer *LNCS 1578*, 1999, 227-242.

La78. Lamport, L., "Time, clocks, and the ordering of events in a distributed system", *Comm. ACM 21(7)*, 1978, 558-565.

LL90. Lamport, L. and Lynch, N., "Distributed computing: Models and methods" in J. van Leeuwen (ed.), North-Holland *Handbook of Theoretical Computer Science, Volume B*, 1990, 1157-1199.

LRT92. Lodaya, K., Ramanujam, R. and Thiagarajan, P.S., "Temporal logics for communicating sequential agents", *Int. Journal of FOCS 3(2)*, 1992, 117-159.

Lyn96. Lynch, N., *Distributed algorithms*, Morgan Kaufmann, 1996.

MP95. Manna, Z. and Pnueli, A., *Temporal verification of reactive systems*, vol 1: Specification, vol 2: Verification, Springer, 1995.

MS99. Montangero, C., and Semini, L., "Composing specifications for coordination", Springer *LNCS 1594*, 1999.

Ra96. Ramanujam, R., "Locally linear time temporal logic", *Proc. LICS*, 1996, 118-127.

Extended Notions of Security for Multicast Public Key Cryptosystems

Olivier Baudron, David Pointcheval, and Jacques Stern

École Normale Supérieure, LIENS
45, rue d'Ulm
F-75230 Paris Cedex 05, France
{Olivier.Baudron,David.Pointcheval,Jacques.Stern}@ens.fr

Abstract. In this paper we introduce two notions of security: multi-user indistinguishability and multi-user non-malleability. We believe that they encompass the correct requirements for public key encryption schemes in the context of multicast communications. A precise and non-trivial analysis proves that they are equivalent to the former single-user notions, provided the number of participants is polynomial. We also introduce a new definition for non-malleability which is simpler than those currently in use. We believe that our results are of practical significance: especially they support the use of PKCS#1 v.2 based on OAEP in the multicast setting.
Keywords: Multicast encryption, semantic security, non-malleability.

1 Introduction

1.1 Motivation

With the growth of wide area networks, cryptographic tools often have to coexist and perform related computations. This may raise new security concerns. For example, broadcast encryption has been the subject of several specific attacks, notably directed against low-exponent RSA [20]. Basically, if e is the common public exponent, then e encryptions of a given message under different public keys lead to an easy recovery of the plaintext. Further results by Håstad [14,22] and Coppersmith [6,7] proved that "time stamp" variants of broadcast, attaching time to the message before encryption, can be successfully cryptanalyzed with e encrypted messages. So far, most known attacks against RSA assume that related plaintexts have been encrypted to different destinations, which enables an eavesdropper to take advantage of the strong dependences between the RSA permutations, although each one is individually one-way.

Despite these attacks, RSA with small exponents is the de facto standard and multicast encryption is performed in many products by encapsulating a symmetric key within several RSA encryptions together with side data which are specific to each receiver. This is precisely the context that we wish to address and we believe that the related security issues needed to be cleared up in order to ensure confidence in standard designs that allow multicast encryption such as PKCS#1. Thus, albeit technical, our research is of practical significance.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 499–511, 2000.
© Springer-Verlag Berlin Heidelberg 2000

1.2 Notions of Security for Encryption

In this paper, we wish to propose notions of security that adequately prevent the attacks just mentioned. Usually, a security level is analyzed in terms of the goal and power of an adversary. The ultimate goal that can be achieved is called *invertibility*: given a public key and an encryption of m, retrieve the whole plaintext m. The RSA assumption implies that the basic RSA encryption scheme is non-invertible. As shown in the above example, the related notion dramatically collapses in a broadcast attack. In a different context, stronger notions of security, have been proposed. Goldwasser and Micali define *semantic security* [13] (also called *indistinguishability*) as the inability for an adversary to distinguish encryptions of two plaintexts. This requires probabilistic encryption, where each plaintext has many corresponding ciphertexts, depending on a random parameter. Recent successful attacks against RSA-like cryptosystems [8] based on known plaintext relations stresses the need for proven schemes achieving semantic security.

Surprisingly, the relationship between broadcast attacks and the improved notions of security has not been the subject of specific research, even if known cryptanalyses seem to fail against semantic security. The motivation of this paper is to investigate whether semantic security, contrary to invertibility, is robust in scenarii involving a general notion of multicast. Our first result gives a positive answer: if one can gain a bit of information by considering a specific set of multicast encrypted messages, then at least one scheme used for encryption is not semantically secure. The proof relies on the hybrid technique and is conceptually simple. It is an independant work of Bellare, Boldyreva and Micali who adressed the same problem [1].

Next, we develop a similar analysis with the notion of *non-malleability*, introduced by Dolev, Dwork and Naor [11]. Informally, the notion asserts that, given a ciphertext, it should be impossible to generate a different ciphertext so that the respective plaintexts are related. The problem of encrypted bids is a famous situation where an eavesdropper may try to under-bid a ciphertext of an unknown amount s, without learning anything about s. This is precisely what non-malleability tries to prevent. A broadcast scenario may be envisioned where several recipients collect the bids over a network. The multicast notion requires that the view of many encrypted messages under different public keys gives no advantage in producing the encryption of a related plaintext. Again, we prove that our new definition of multi-user non-malleability is equivalent to the former single-user notion: no broadcast attack can be performed against a non-malleable scheme. Here, the reduction is definitely much harder to obtain. Due to the complex nature of the definitions, involving auxiliary distributions of plaintexts and binary relations, both issued by the attacker, our previous natural reduction cannot be applied. The major technical point of the proof relies on a lemma embedding any distribution into the product of a 2 element-distribution which leads to a simpler definition of non-malleability. We think that this lemma may be of independent interest to cryptographers.

We now discuss the notion of security in terms of the adversary's power. Usually, an attacker is a probabilistic polynomial time Turing machine running in two stages. Firstly, given a public key, it achieves a precomputation stage and halts. From the output data, a challenge is randomly encrypted and given to the attacker which performs a second stage of computation. The polynomial strength of the attacker may be increased by providing him access to a decryption oracle. Whether the oracle is accessible during the first stage only or during whole computation leads to three different scenarii. Under a *chosen-plaintext attack* the adversary can obtain ciphertexts of his choice, which is meaningless in the context of public key encryption. Under *chosen ciphertext attack* [17], the adversary is allowed to use a decryption oracle during the precomputation stage only. Lastly, under *adaptive chosen ciphertext attack* [19], the adversary is allowed to use a decryption oracle during whole algorithm, with the trivial restriction that the challenge cannot be asked to the oracle. The latter is the ideal candidate that one should consider in order to provide the best arguments for security. In our paper, whenever a theorem is stated, it is assumed that one of the three contexts given above has been fixed and hence no decryption oracle is mentioned; potential oracles are preferably viewed as internal parts of the attacker.

1.3 Outline of the Paper

The rest of the paper is organized as follows. Section 2 gives common definitions and notations for encryption and probabilities. Sections 3 and 4 contain our analysis of semantic security (which we call indistinguishability) and non-malleability. Both introduce definitions of these notions in the context of multicast. The conclusion follows in section 5.

2 Definitions and Notations

A public key encryption scheme Π is a triplet $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ consisting of three probabilistic polynomial time algorithms.

- $\mathcal{K}$ is the *key generation algorithm* which, given a security parameter k (usually viewed as a unary input 1^k) produces from its random source ω a pair (pk, sk) of public and secret keys.
- $\mathcal{E}$ is the probabilistic *encryption algorithm* which, given the security parameter k, defines a message space $\mathcal{M}$ such that: for each string x from $\mathcal{M}$, and for each valid public key pk, $\mathcal{E}_{pk}(x)$ is a string y, called the *encryption* of x under pk.
- $\mathcal{D}$ is the (deterministic) *decryption algorithm*. It is required that for every message x in $\mathcal{M}$ and for every pair (pk, sk) output by $\mathcal{K}$, $\mathcal{D}_{sk}(\mathcal{E}_{pk}(x)) = x$. In all other cases, the output of $\mathcal{D}$ is any element of $\mathcal{M} \cup \{\perp\}$. A ciphertext whose decryption is $\perp$ is said to be *invalid*.

A real-valued function $f(n)$ is *negligible* if for any integer k, $|f(n)| < n^{-k}$ for sufficiently large n.

Given a distribution δ over a finite space Ω, we let $\Pr_\delta[E]$ be the probability of an event E. When δ is omitted, it is implicitly assumed that δ is the uniform distribution. The *support* of δ is the set of elements from Ω whose probability is non zero. Often, a random variable is conveniently defined by the output distribution of a probabilistic Turing machine. We let $y \leftarrow TM(x)$ be the result y by running TM on input x and random source ω. If S is a finite set then $y \leftarrow S$ is the operation of picking an element uniformly in S.

When considering several encryption schemes $\Pi_1, .., \Pi_n$ and their related algorithms, we will denote by $\mathcal{K}^n$, $\mathcal{E}^n$ and $\mathcal{D}^n$ the algorithms that given an input vector of n adequate data, output a vector of dimension n whose distribution is given by the product of the output distributions of $\mathcal{K}_1 \times .. \times \mathcal{K}_n$, $\mathcal{E}_1 \times .. \times \mathcal{E}_n$ and $\mathcal{D}_1 \times .. \times \mathcal{D}_n$ respectively. We insist that all encryption schemes need not be identical.

Our multicast notion enlarges the intuitive definition of broadcast when a unique plaintext is encrypted. In this paper, we consider a multicast communication as a set of encryptions of suitably related plaintexts under different public keys. For example the reader might consider messages containing the name of the recipient followed by a possibly common text. Formally, a broadcast distribution of plaintexts is any *diagonal* distribution whose support is in $\mathcal{M}^n$ whereas a multicast distribution of plaintexts is any distribution whose support is in $\mathcal{M}^n$.

3 Indistinguishability

3.1 Single-User Encryption Schemes

Secure encryption should preserve privacy even in the critical context where the messages are taken from a small set of plaintexts: it should be impossible for an eavesdropper to distinguish encryptions of distinct values. Such a requirement is captured by the notion of indistinguishability, also known as semantic security [13,15]. Examples, secure against chosen plaintext attack, include El Gamal [12] (based on the decisional Diffie-Hellman assumption [10]), Naccache-Stern [16] (based on higher residues) and Okamoto-Uchiyama [18] (based on factorization). Our definition exactly follows [2] and uses the same notations. Indistinguishability is defined by the advantage of an adversary $A = (A_1, A_2)$ performing a sequence of two algorithms.

In a first step, algorithm A_1 is run on input of the public key pk and outputs two plaintexts messages x^0 and x^1 plus a string s encoding information to be handled to A_2. Next a message from $\{x^0, x^1\}$ is chosen at random and encrypted into a challenge ciphertext y. In a second step, A_2 is given the input (y, s) and has to guess the bit of the plaintext being encrypted. The advantage of A is measured by the probability that it outputs the correct bit of the challenge. The scheme is indistinguishable if no adversary obtains an advantage significantly greater than one would obtain by flipping a coin. The formal definition follows:

Definition 1. *Single-user indistinguishability.*
Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme with a security parameter k and let $A = (A_1, A_2)$ be an adversary. For $k \in \mathbb{N}$, we define the advantage:

$$\mathsf{Adv}_{A,\Pi}(k) = 2\Pr\Big[(pk, sk) \leftarrow \mathcal{K}(1^k);\ (x^0, x^1, s) \leftarrow A_1(pk);\ b \leftarrow \{0,1\};\\ y \leftarrow \mathcal{E}_{pk}(x^b)\ :\ A_2(s, y) = b\Big] - 1$$

We say that Π is single-user indistinguishable (S-IND) if for every polynomial time adversary A, $\mathsf{Adv}_{A,\Pi}(k)$ is negligible.

3.2 Multicast Encryption Schemes

In the context of multicast, the usual notion of indistinguishability does not, by itself, guarantee that no bit of information is leaked when putting together the encryptions of related messages under different public keys. Our definition captures this stronger notion of security by giving the adversary the ability to choose two vectors of plaintexts whose coordinates are plaintext messages possibly related or even identical. Next, one of the two vectors is chosen at random and is encrypted coordinatewise with the different public keys. The final goal of the adversary is to guess which one was encrypted. This is easily done if a boolean function distinguishes the two vectors of plaintexts and is computable from the encrypted data. Again our formal definition is in terms of the advantage of an adversary playing the game just given. In the following, underlined variables denote vectors of size n; the ith coordinate refers to the ith cryptosystem.

Definition 2. *Multi-user indistinguishability.*
Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme with a security parameter k and let $A = (A_1, A_2)$ be an adversary. For $k, n \in \mathbb{N}$, we define the advantage:

$$\mathsf{Adv}_{A,\Pi}(k, n) = 2\Pr\Big[(\underline{pk}, \underline{sk}) \leftarrow \mathcal{K}^n(1^k);\ (\underline{x}^0, \underline{x}^1, s) \leftarrow A_1(\underline{pk});\ b \leftarrow \{0,1\};\\ \underline{y} \leftarrow \mathcal{E}_{\underline{pk}}(\underline{x}^b)\ :\ A_2(s, \underline{y}) = b\Big] - 1$$

We say that Π is multi-user indistinguishable (M-IND) if for every polynomial time adversary A, $\mathsf{Adv}_{A,\Pi}(k, n)$ is negligible.

3.3 Results

As expected, any multi-user indistinguishable encryption scheme Π is also single-user indistinguishable. Indeed, if an adversary distinguishes $\mathcal{E}_{pk}(m^0)$ from $\mathcal{E}_{pk}(m^1)$ then it obviously distinguishes two encrypted vectors whose first coordinate is the encryption of m^0 and m^1 under the public key pk. Also note that the usual definition of (single-user) indistinguishability, expressed in [2], is the particular case of multi-user indistinguishability where $n = 1$. The following result achieves equivalence.

Theorem 1. *S-IND⇒M-IND.*
If encryption scheme Π is single-user indistinguishable, then it is multi-user indistinguishable.

Proof. Let A be an adversary attacking Π in the sense of M-IND. We build n adversaries $B_i = (B_{i,1}, B_{i,2})_{1 \leq i \leq n}$, as follows:

$\mathsf{Algorithm} B_{i,1}(pk_i)$:
$\quad \underline{pk} \leftarrow (pk_1, .., pk_i, .., pk_n)$
$\quad (\underline{x}^0, \underline{x}^1, s) \leftarrow A_1(\underline{pk})$
$\quad \mathsf{return}(x_i^0, x_i^1, s)$

$\mathsf{Algorithm} B_{i,2}(y_i, s)$:
$\quad b' \leftarrow \{0,1\}$
$\quad \underline{y} \leftarrow (y_1, .., y_i, .., y_n)$ with $y_j = \mathcal{E}_{pk_j}(x_j^{b'})$ if $j < i$
$\qquad\qquad\qquad\qquad\qquad\qquad\quad y_j = \mathcal{E}_{pk_j}(x_j^{\bar{b}'})$ if $j > i$
$\quad b'' \leftarrow A_2(\underline{y}, s)$
$\quad \mathsf{return}\ b''$

In a first step $B_{i,1}$ extends pk_i to a vector of public keys $\underline{pk}$, using $(n-1)$ times the algorithm $\mathcal{K}$. Then A_1 is run with the input $\underline{pk}$. The ith pair of plaintext messages output by A_1 is returned, which completes the first part of the algorithm. We note b the unknown bit of the challenge, i.e. $y_i = \mathcal{E}_{pk_i}(x_i^b)$. In a second step, $B_{i,2}$ extends its input y_i to a hybrid vector $\underline{y}$: the first coordinates of $\underline{y}$ come from the encryption of $\underline{x}^{b'}$ whereas the last coordinates of $\underline{y}$ come from the encryption of $\underline{x}^{\bar{b}'}$. Bit b'' output by running A_2 on $\underline{y}$ is returned as an answer to the challenge.

We now compute the advantage of B_i for $\underline{pk}$, $\underline{x}^0$, $\underline{x}^1$ and s fixed. Let d be a random bit and let $\Pr_i$ (respectively $\Pr'_i$) be the probability that the initial adversary A_2 successfully guesses the plaintext of the left (respectively right) part of a hybrid ciphertext formed with i coordinates from x^d followed by $(n-i)$ coordinates from $x^{\bar{d}}$:

$$\Pr_i = \Pr\left[d \leftarrow \{0,1\};\ \underline{c} \leftarrow \mathcal{E}_{\underline{pk}}(a_1^d, .., a_i^d, a_{i+1}^{\bar{d}}, .., a_n^{\bar{d}});\ d' \leftarrow A_2(\underline{c}, s)\ :\ d' = d\right]$$
$$\Pr'_i = \Pr\left[d \leftarrow \{0,1\};\ \underline{c} \leftarrow \mathcal{E}_{\underline{pk}}(a_1^d, .., a_i^d, a_{i+1}^{\bar{d}}, .., a_n^{\bar{d}});\ d' \leftarrow A_2(\underline{c}, s)\ :\ d' \neq d\right]$$

Note that,

$$\Pr_i + \Pr'_i = 1 \tag{1}$$

We apply Bayes' theorem, considering the value of the bit b' randomly chosen in the algorithm $B_{i,2}$:

$$\begin{aligned}
&\Pr\left[b \leftarrow \{0,1\};\ y_i \leftarrow \mathcal{E}_{pk_i}(x_i^b);\ b'' \leftarrow B_{i,2}(y_i, s)\ :\ b'' = b\right] \\
&= \tfrac{1}{2}\Pr\left[b \leftarrow \{0,1\};\ y_i \leftarrow \mathcal{E}_{pk_i}(x_i^b);\ b'' \leftarrow B_{i,2}(y_i, s)\ :\ b'' = b \mid b' = b\right] \\
&\quad + \tfrac{1}{2}\Pr\left[b \leftarrow \{0,1\};\ y_i \leftarrow \mathcal{E}_{pk_i}(x_i^b);\ b'' \leftarrow B_{i,2}(y_i, s)\ :\ b'' = b \mid b' \neq b\right] \\
&= \tfrac{1}{2}\Pr_i + \tfrac{1}{2}\Pr'_{i-1}
\end{aligned} \tag{2}$$

It follows from (1) and (2) that the advantage of B_i is:

$$\mathsf{Adv}_{B_i,\Pi} = 2\left(\tfrac{1}{2}\Pr_i + \tfrac{1}{2}\Pr'_{i-1}\right) - 1 = \Pr_i - \Pr_{i-1}$$

Middle terms cancel in the sum, so that:

$$\sum_{i=1}^{n} \mathsf{Adv}_{B_i,\Pi} = \Pr_n - \Pr_0 = \mathsf{Adv}_{A,\Pi}$$

Consequently, if i is uniformly chosen at random in $\{1,..,n\}$, we obtain a reduction from a multi-distinguisher attacker A with advantage ϵ, to a single-distinguisher attacker B with advantage ϵ/n. □

4 Non-malleability

4.1 Single-User Non-malleability

The notion of non-malleability was introduced in [11] and formalized in a different manner in [2]. The main idea is that, given an encrypted message y, an adversary is unable to output a ciphertext y' whose decryption is related to the decryption of y. More precisely, this goes along an interactive experiment with an adversary $A = (A_1, A_2)$ which is described below.

The Turing machine A_1 is run with input of a public key pk and outputs the description of a probabilistic polynomial time Turing machine M, and a string s for further computation. The output of M defines a distribution of plaintext messages whose support is a set $|M| \subset \mathcal{M}$. In the following M refers to the Turing machine as well as its output distribution. Then a message x is randomly chosen by running M and its encryption is given to A_2. The goal of A_2 is to output a binary relation R over $|M| \times \mathcal{M}$ and a ciphertext $y' \neq y$ whose decryption x' is related to x according to R. The scheme is non-malleable if for any adversary the probability that $R(x, x')$ holds is not significantly better than the probability that $R(\tilde{x}, x')$ for a random $\tilde{x}$ from M.

For notational convenience we have simplified the definition given in [2]. In the original paper, the goal of the adversary was to output a vector $\mathbf{y}'$ of $t-1$ ciphertexts related to y according to a relation R of arity t. In this case, it is required that no coordinate of $\mathbf{y}'$ is equal to y. It was also proven that both definitions were not equivalent. The former could not be reduced to the latter. In the rest of our paper we will only represent elements y' with one coordinate so that no confusion arises with vectors from the broadcast notation. But one can also build a similar theory of multi-user non-malleability for relations of arity t by considering the modified ciphertext as a vector of ciphertext vectors $\underline{\mathbf{y}}'$ and an appropriate binary relation over $|M| \times \mathcal{M}^{n\times(t-1)}$.

Recently, it was shown by Bellare and Sahai [4] that non-malleability (in any attack model) was equivalent to indistinguishability where the adversary gets the additional power of "parallel ciphertext attack" (i.e. non adaptive ciphertext attack after seeing the challenge encryption). Consequently, our first result may apply to this notion. However, we followed the standard definition of non-malleability and proved it may be simplified.

4.2 Multi-user Non-malleability

Scenarii where it is unclear whether single-non-malleability is enough to ensure a satisfactory notion of security can be envisioned: for example, the view of different encryptions under several public keys might give the opportunity for an adversary to flip one of the encrypted message into its opposite. It is also not clear that encrypted messages sent to different users may not be exchanged. Thus, if one wishes to cover the standard context of multicast it is natural to give an extended notion of security for non-malleability which we now undertake.

The adversary is given n public keys and outputs a probabilistic polynomial time Turing machine M plus a string s. By running M on a random source we require that its output defines a distribution of plaintext messages whose support $|M|$ is in $\mathcal{M}^n$. Then, a vector $\underline{x}$ is randomly chosen by running M, and its coordinatewise encryption according to the different public keys is given to A_2. The goal of A_2 is to output a vector of ciphertexts $\underline{y}'$ and a relation R over $|M| \times \mathcal{M}^n$. A is successful if R relates the corresponding decrypted messages. The formal definition is given below.

Remark. The exact support $|M|$ of M may not be computable in polynomial time. It is therefore only required that the relation R is defined on a subset of $\mathcal{M}^n \times \mathcal{M}^n$ and covers $|M| \times \mathcal{M}^n$.

Definition 3. *Multi-user non-malleability.*
Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be an encryption scheme with security parameter k and let $A = (A_1, A_2)$ be an adversary. For $k, n \in \mathbb{N}$, we define the advantage:

$$\mathsf{Adv}_{A,\Pi}(k,n) = \left|\mathsf{Succ}_{A,\Pi}(k,n) - \mathsf{Succ}_{A,\Pi,\$}(k,n)\right|,$$

where

$$\mathsf{Succ}_{A,\Pi}(k,n) = \Pr\Big[\ (\underline{pk}, \underline{sk}) \leftarrow \mathcal{K}^n(1^k);\ (M,s) \leftarrow A_1(\underline{pk});\ \underline{x} \leftarrow M;\ \underline{y} \leftarrow \mathcal{E}_{\underline{pk}}(\underline{x});\ (R, \underline{y}') \leftarrow A_2(M,s,\underline{y});\ \underline{x}' \leftarrow \mathcal{D}_{\underline{sk}}(\underline{y}')\ :\ \perp \notin \underline{x}' \wedge R(\underline{x}, \underline{x}')\Big]$$

$$\mathsf{Succ}_{A,\Pi,\$}(k,n) = \Pr\Big[\ (\underline{pk}, \underline{sk}) \leftarrow \mathcal{K}^n(1^k);\ (M,s) \leftarrow A_1(\underline{pk});\ \underline{x}, \tilde{\underline{x}} \leftarrow M;\ \underline{y} \leftarrow \mathcal{E}_{\underline{pk}}(\underline{x});\ (R, \underline{y}') \leftarrow A_2(M,s,\underline{y});\ \underline{x}' \leftarrow \mathcal{D}_{\underline{sk}}(\underline{y}')\ :\ \perp \notin \underline{x}' \wedge R(\tilde{\underline{x}}', \underline{x}')\Big]$$

$$\text{with } \tilde{x}'_i = \begin{cases} x_i \text{ if } & y'_i = y_i \\ \tilde{x}_i \text{ if } & y'_i \neq y_i \end{cases}, \text{ for each } i \text{ in } \{1, .., n\}$$

We say that Π is multi-user non-malleable (M-NM) if for every polynomial time adversary A whose output is a distribution of plaintexts M and a relation R both computable in polynomial time then $\mathsf{Adv}_{A,\Pi}$ is negligible.

The motivation to introduce a new variable $\tilde{\underline{x}}'$ was to restrict the domain of the random variable $\tilde{\underline{x}}$ for the coordinates left unchanged by A_2. This is the condition in dimension n of the requirement $y' \neq y$ in dimension 1, defined in [2]. This rule makes the adversary gain no advantage in partially copying a vector

of ciphertexts and outputting a relation whose value is true on domains of the form $((x_0, .., *), (x_0, .., *))$.

The usual notion of (single-user) non-malleability is the particular case where n is fixed to 1.

4.3 Results

The next result is the main technical achievement of our paper and leads to a simplified definition of non-malleability. It claims that the distribution of plaintexts M can be restricted to an atomic form.

Lemma 1. *Atomic non-malleability.*
Let Π be an encryption scheme and let A be an adversary attacking Π in the sense of M-NM. Then there exists another adversary B attacking Π, in the sense of M-NM such that the distribution of plaintexts that B outputs is always a uniform distribution of two vectors of plaintexts. Moreover, the running time of B is that of A plus the running time of the Turing machine M output by A.

Proof. The adversary $B = (B_1, B_2)$ is defined as follows:

Algorithm $B_1(\underline{pk})$ $(M, s) \leftarrow A_1(\underline{pk})$ $\underline{a}^0 \leftarrow M;\ \underline{a}^1 \leftarrow M$ return $(\{\underline{a}^0, \underline{a}^1\}, s)$	Algorithm $B_2(\underline{y}, s)$ $(R, \underline{y}') \leftarrow A_2(\underline{y}, s)$ return $(R, \underline{y}')$

Here the description of B_2 is identical to A_2 except that the relation R is restricted to the set $\{\underline{a}^0, \underline{a}^1\} \times \mathcal{M}^n$ instead of $M \times \mathcal{M}^n$. We first claim that the input distribution of the ciphertexts is the same for A_2 and B_2. Indeed, using Bayes' theorem and since $\underline{x}$ has equal probability 1/2 of being $\underline{a}_0$ or $\underline{a}_1$, it results that for all $\underline{X}$ in M:

$$\begin{aligned}
&\Pr\left[\underline{a}^0, \underline{a}^1 \leftarrow M;\ \underline{x} \leftarrow \{\underline{a}^0, \underline{a}^1\}\ :\ \underline{x} = \underline{X}\right] \\
&\quad = \tfrac{1}{2}\Pr\left[\underline{a}^0 \leftarrow M\ :\ \underline{a}^0 = \underline{X}\right] + \tfrac{1}{2}\Pr[\underline{a}^1 \leftarrow M\ :\ \underline{a}^1 = \underline{X}] \\
&\quad = \Pr\left[\underline{x} \leftarrow M\ :\ \underline{x} = \underline{X}\right]
\end{aligned}$$

Consequently, $\mathsf{Succ}_{B,\Pi} = \mathsf{Succ}_{A,\Pi}$. Next, in order to express $\mathsf{Succ}_{B,\Pi,\$}$ we decorelate $\tilde{\underline{x}}$ from $\underline{x}$, considering its two possible values among $\{\underline{a}^0, \underline{a}^1\}$. Using the notations from definition 3, it holds:

$$\begin{aligned}
&\Pr\left[\underline{a}^0, \underline{a}^1 \leftarrow M;\ \underline{x}, \tilde{\underline{x}} \leftarrow \{\underline{a}^0, \underline{a}^1\}\ :\ R(\tilde{\underline{x}}', \underline{x}')\right] \\
&= \tfrac{1}{2}\Pr\left[\underline{a}^0, \underline{a}^1 \leftarrow M;\ \underline{x}, \tilde{\underline{x}} \leftarrow \{\underline{a}^0, \underline{a}^1\}\ :\ R(\tilde{\underline{x}}', \underline{x}') \mid \tilde{\underline{x}} = \underline{x}\right] \\
&\quad + \tfrac{1}{2}\Pr\left[\underline{a}^0, \underline{a}^1 \leftarrow M;\ \underline{x}, \tilde{\underline{x}} \leftarrow \{\underline{a}^0, \underline{a}^1\}\ :\ R(\tilde{\underline{x}}', \underline{x}') \mid \tilde{\underline{x}} \neq \underline{x}\right] \\
&= \tfrac{1}{2}\Pr\left[\underline{a}^0, \underline{a}^1 \leftarrow M;\ \underline{x} \leftarrow \{\underline{a}^0, \underline{a}^1\}\ :\ R(\underline{x}, \underline{x}')\right] \\
&\quad + \tfrac{1}{2}\Pr\left[\tilde{\underline{a}}^0, \underline{a}^1 \leftarrow M\ :\ R(\tilde{\underline{a}}^{0\prime}, \underline{a}^{1\prime})\right]
\end{aligned}$$

So, $\mathsf{Succ}_{B,\Pi,\$} = \frac{1}{2}\mathsf{Succ}_{B,\Pi} + \frac{1}{2}\mathsf{Succ}_{A,\Pi,\$}$ and $\mathsf{Adv}_{B,\Pi} = \mathsf{Succ}_{B,\Pi} - \mathsf{Succ}_{B,\Pi,\$} = \frac{1}{2}\mathsf{Succ}_{B,\Pi} - \frac{1}{2}\mathsf{Succ}_{A,\Pi,\$}$. With the previous result, we conclude

$$\mathsf{Adv}_{B,\Pi} = \frac{1}{2}\mathsf{Adv}_{A,\Pi}$$

□

It is easily seen that the definition of single-user non-malleability is the restricted case of the multi-user non-malleability for $n = 1$. The equivalence follows from the next result.

Theorem 2. *S-NM⇒M-NM. If encryption scheme Π is single-user non-malleable, then it is multi-user non-malleable.*

Proof. Let $A = (A_1, A_2)$ be an adversary attacking Π in the sense of multi-user non-malleability with an advantage ϵ. Without loss of generality, as was shown in Lemma 1, we assume that A_1 outputs a uniform distribution M of two plaintext vectors $\underline{a}_0$ and $\underline{a}_1$. We will build n Turing machine $B_1, .., B_n$ attacking Π in the sense of single-user non-malleability. For any $i \in \{1, .., n\}$, the description of $B_i = (B_{i,1}, B_{i,2})$ is as follows:

Algorithm $B_{i,1}(pk_i)$:
$\quad \underline{pk} \leftarrow (pk_1, .., pk_i, .., pk_n)$
$\quad (M, s) \leftarrow A_1(\underline{pk})$
$\quad$ return $M_i = \{a_i^0, a_i^1\}$

Algorithm $B_{i,2}(c_i, s)$:
$\quad b' \leftarrow \{0, 1\}$
$\quad \underline{c} \leftarrow (c_1, .., c_i, .., c_n)$ with $c_j = \mathcal{E}_{pk_j}(a_j^{b'})$ if $j < i$
$\qquad\qquad\qquad\qquad\qquad\qquad c_j = \mathcal{E}_{pk_j}(a_j^{b'})$ if $j > i$
$\quad (\underline{c}', R) \leftarrow A_2(\underline{c}, s)$
$\quad R_i(a_i^k, u) \iff R(\underline{a}^k, \underline{v})$ with $v_i = u$
$\qquad\qquad\qquad\qquad\qquad\qquad v_j = \mathcal{D}_{sk_j}(c'_j)$ if $j \neq i$
$\quad$ return (c'_i, R_i)

As in the previous construction, the first part of the algorithm extends the input pk_i into a vector $\underline{pk}$ and calls the attacker A_1 on this data. Without loss of generality, as was shown in Lemma 1, A_1 outputs a distribution M of two plaintexts $\underline{a}_0$ and $\underline{a}_1$. Then both i^{th} coordinates are returned. The algorithm $B_{i,2}$ takes as input the ciphertext c_i of a plaintext a_i^b where b is an unknown bit. We focus on the way the binary relation R_i over $\{a_i^0, a_i^1\} \times \mathcal{M}$ is built from the initial relation R over $\{\underline{a}_0, \underline{a}_1\} \times \mathcal{M}^n$. Since the expression of the advantage of A only depends on the decryption of $\underline{c}$, we let the ith coordinate free and fix the others to the decrypted coordinates of $\underline{c}$ thanks to the knowledge of the related secret keys. Thus R_i is the section of R on this particular sub-space. Note that, the exact definition of R_i may be ambiguous in the case where $a_i^0 = a_i^1$ and $\underline{a}^0 \neq \underline{a}^1$. Here, it is clear that any attacker (even infinitely powerful) obtains a null advantage

since the encryption of a_i^b is perfectly independent of the bit b. Thus in this specific case, the definition of R_i has little importance, and for convenience, it is defined by choosing b randomly so that the following computations remain true.

We now fix $\underline{pk}$, $\underline{a}_0$ and $\underline{a}_1$. The main goal is to analyze the behavior of the adversary A_2 when its input is a hybrid vector of ciphertexts from $\underline{a}_0$ and $\underline{a}_1$. Let $\Pr_i$ (respectively $\Pr'_i$) be the probability that A_2 successfully outputs a ciphertext related to the first (respectively last) part of the initial hybrid plaintext.

$$\Pr_i = \Pr\left[b \leftarrow \{0,1\};\ \underline{c} \leftarrow \mathcal{E}_{\underline{pk}}(a_1^b,..,a_i^b,a_{i+1}^{\bar{b}},..,a_n^{\bar{b}}); (\underline{c}',R) \leftarrow A_2(\underline{c},s) : R(\underline{a}^b,\mathcal{D}_{\underline{sk}}(\underline{c}'))\right]$$
$$\Pr'_i = \Pr\left[b \leftarrow \{0,1\};\ \underline{c} \leftarrow \mathcal{E}_{\underline{pk}}(a_1^b,..,a_i^b,a_{i+1}^{\bar{b}},..,a_n^{\bar{b}}); (\underline{c}',R) \leftarrow A_2(\underline{c},s) : R(\underline{a}^{\bar{b}},\mathcal{D}_{\underline{sk}}(\underline{c}'))\right]$$

Remark: If $a_i^0 = a_i^1$ then a_i^b can be linked identically to the left part or the right part of the hybrid, hence $\Pr_i = \Pr_{i-1}$ and $\Pr'_i = \Pr'_{i-1}$.

It follows from the above definitions that $\Pr_n = \Pr'_0$ and $\Pr'_n = \Pr_0$.
The success of the attacker B_i is:

$$\begin{aligned}
&\mathsf{Succ}_{B_i,\Pi}\\
&= \Pr\left[b,b' \leftarrow \{0,1\};\ \underline{c} \leftarrow (c_1,..,c_i,..,c_n);\ (\underline{c}',R) \leftarrow A_2(\underline{c},s)\ :\ R(\underline{a}^b,\mathcal{D}_{\underline{pk}}(\underline{c}'))\right]\\
&= \tfrac{1}{2}\Pr\left[b,b' \leftarrow \{0,1\};\ \underline{c} \leftarrow (c_1,..,c_n);\ (\underline{c}',R) \leftarrow A_2(\underline{c},s)\ :\ R(\underline{a}^b,\mathcal{D}_{\underline{pk}}(\underline{c}')) \mid b' = b\right]\\
&\quad + \tfrac{1}{2}\Pr\left[b,b' \leftarrow \{0,1\};\ \underline{c} \leftarrow (c_1,..,c_n);\ (\underline{c}',R) \leftarrow A_2(\underline{c},s)\ :\ R(\underline{a}^b,\mathcal{D}_{\underline{pk}}(\underline{c}')) \mid b' \neq b\right]\\
&= \tfrac{1}{2}\Pr_i + \tfrac{1}{2}\Pr'_{i-1}
\end{aligned}$$

The average success Succ is obtained by considering the four possible values of the B-bit b' and the random bit $\tilde{b}$ relatively to the challenge bit b. Since b shares the vector $\underline{c}$ into a left part of $i-1$ encrypted coordinates from b' and a right part of $(n-1-i)$ encrypted coordinates from $\bar{b}'$, whether b is equal to b' or $\bar{b}'$ leads to an hybrid vector $\underline{c}$ whose frontier is at position i or $i-1$. In each case, whether the random bit $\tilde{b}$ is the left or the right part of the hybrid vector $\underline{c}$, leads to one of the expressions Pr or Pr$'$.

Let the distribution: $\delta = \left\{b,b',\tilde{b} \leftarrow \{0,1\};\ \underline{c} \leftarrow (c_1,..,c_i,..,c_n);\ (\underline{c}',R) \leftarrow A_2(\underline{c},s)\right\}$.

$$\begin{aligned}
&\mathsf{Succ}_{B_i,\Pi,\$}\\
&= \Pr_\delta\left[R(\underline{a}^{\tilde{b}},\mathcal{D}_{\underline{pk}}(\underline{c}'))\right]\\
&= \tfrac{1}{4}\Pr_\delta\left[R(\underline{a}^{\tilde{b}},\mathcal{D}_{\underline{pk}}(\underline{c}')) \mid \tilde{b}=b \wedge b'=b\right] + \tfrac{1}{4}\Pr_\delta\left[R(\underline{a}^{\tilde{b}},\mathcal{D}_{\underline{pk}}(\underline{c}')) \mid \tilde{b}=b \wedge b'\neq b\right]\\
&\quad + \tfrac{1}{4}\Pr_\delta\left[R(\underline{a}^{\tilde{b}},\mathcal{D}_{\underline{pk}}(\underline{c}')) \mid \tilde{b}\neq b \wedge b'=b\right] + \tfrac{1}{4}\Pr_\delta\left[R(\underline{a}^{\tilde{b}},\mathcal{D}_{\underline{pk}}(\underline{c}')) \mid \tilde{b}\neq b \wedge b'\neq b\right]\\
&= \tfrac{1}{4}\Pr_i + \tfrac{1}{4}\Pr'_{i-1} + \tfrac{1}{4}\Pr'_i + \tfrac{1}{4}\Pr'_{i-1}
\end{aligned}$$

It follows that the advantage of B_i is:

$$\mathsf{Adv}_{B_i} = \mathsf{Succ}_{B_i,\Pi} - \mathsf{Succ}_{B_i,\Pi,\$} = \tfrac{1}{4}\Pr_i + \tfrac{1}{4}\Pr'_{i-1} - \tfrac{1}{4}\Pr'_i - \tfrac{1}{4}\Pr_{i-1}$$

Remark: if $a_i^0 = a_i^1$ then from the previous remark $Adv_{B_i} = 0$ as expected.

Finally the sum is:

$$\sum_{i=1}^{n} \mathsf{Adv}_{B_i} = \tfrac{1}{4}(\mathrm{Pr}_n + \mathrm{Pr}'_0 - \mathrm{Pr}'_n - \mathrm{Pr}_0) = \tfrac{1}{2}(\mathrm{Pr}_n - \mathrm{Pr}_0) = \mathsf{Adv}_A$$

Thus, if i is randomly choosen in the set $\{1, .., n\}$, one obtains a reduction from a global adversary with advantage ϵ to an adversary with advantage ϵ/n against a single cryptosystem. □

Consequences of the results. In the case of adaptive chosen ciphertext attacks, it was proved by Bellare *et al.* [2] that both notions of indistinguishability and non-malleability are equivalent, and hence are also equivalent to the multi-user notions of security. Thus, our results show that some recent encryption schemes achieve a high level of multicast security requirement. In the random oracle model, one can mention the RSA-base OAEP [3] from Bellare and Rogaway. It was recently adopted as a standard of encryption in the PKCS#1 [21,5] specifications. In the standard model of proofs, only the Cramer-Shoup scheme [9] achieves proven security and practical effectiveness. Finally, we point out some practical and straightforward applications of multi-user secure encryption. This includes pay-per-view television, where a part of the bandwith is used to broadcast encrypted keys to each user. Secure electronic mail such as PGP is also given better confidence especially when adressing several recipients. One may also envision secure election protocols with a large number of independent authorities generally resulting in many related encrypted plaintexts. Lastly, multiparty computations usually use the assumption of a broadcast channel and thus should benefit from our multicast notions of secutity.

5 Conclusion

We have extended the applicability of two powerful notions of security: indistinguishability and non-malleability. Every known attack is now covered by our new multicast security definitions. Furthermore, the reductions that we have shown have linear coefficients in the number of users. As a consequence, we believe that proven encryptions schemes with common single-user security parameters are ready to be safely spread over the Internet.

Acknoledgments

We thanks the program commitee for their valuable comments.

References

1. M. Bellare, A. Boldyrevaand, and S. Micali. Public-Key Encryption in a Multi-user Setting : Security Proofs and Improvements. In *Eurocrypt '00*, LNCS. Springer-Verlag, 2000.

2. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. In *Crypto '98*, LNCS 1462, pages 26-45. Springer-Verlag, 1998.
3. M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *Eurocrypt '94*, LNCS 950, pages 92-111. Springer-Verlag, 1995.
4. M. Bellare and A. Sahai. Non-Malleable Encryption : Equivalence between Two Notions and an Indistinguishability-Based Characterization. In *Crypto '99*, LNCS 1666, pages 519-536. Springer-Verlag, 1998.
5. D. Bleichenbacher. A Chosen Ciphertext Attack against Protocols based on the RSA Encryption Standard PKCS # 1. In Crypto '98, LNCS 1462, pages 1-12. Springer-Verlag, 1998.
6. D. Coppersmith. Finding a Small Root of a Univariate Modular Equation. In *Eurocrypt '96*, LNCS 1070, pages 155-165. Springer-Verlag, 1996.
7. D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *Journal of Cryptology*, 10:233-260, 1997.
8. D. Coppersmith, M. Franklin, J. Patarin, and M. Reiter. Low-Exponent RSA with Related Messages. In *Eurocrypt '96*, LNCS 1070, pages 1-9. Springer-Verlag, 1996.
9. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure against Adaptive Chosen Ciphertext Attack. In *Crypto '98*, LNCS 1462, pages 13-25. Springer-Verlag, 1998.
10. W. Diffie and M. E. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, volume IT-22, no. 6, pages 644-654, November 1976.
11. D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. In *Proc. of the 23rd STOC. ACM Press*, 1991.
12. T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *IEEE Transactions on Information Theory*, volume IT-31, no. 4, pages 469-472, July 1985.
13. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences*, 28:270-299, 1984.
14. J. Håstad. Solving Simultaneous Modular Equations of Low Degree. SIAM Journal of Computing, 17:336-341, 1988.
15. S. Micali, C. Rackoff, and R. Sloan. The notion of security for probabilistic cryptosystems. *SIAM J. of Computing*, April 1988.
16. D. Naccache and J. Stern. A New Cryptosystem based on Higher Residues. In *Proc. of the 5th CCCS*, pages 59-66. ACM press, 1998.
17. M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In Proc. of the 22nd STOC, pages 427-437. ACM Press, 1990.
18. T. Okamoto and S. Uchiyama. A New Public Key Cryptosystem as Secure as Factoring. In *Eurocrypt '98*, LNCS 1403, pages 308-318. Springer-Verlag, 1998.
19. C. Racko and D. R. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. In Crypto '91, LNCS 576, pages 433-444. Springer-Verlag, 1992.
20. R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. Communications of the ACM, 21(2):120-126, February 1978.
21. RSA Data Security, Inc. Public Key Cryptography Standards – PKCS. Available from `http://www.rsa.com/rsalabs/pubs/PKCS/`.
22. H. Shimizu. On the Improvement of the Håstad Bound. In *1996 IEICE Fall Conference*, Volume A-162, 1996. In Japanese.

One-Round Secure Computation and Secure Autonomous Mobile Agents

(Extended Abstract)

Christian Cachin[1], Jan Camenisch[1], Joe Kilian[2], and Joy Müller[1]

[1] IBM Zurich Research Laboratory, CH-8803 Rüschlikon, Switzerland, {cca,jca,jmu}@zurich.ibm.com

[2] NEC Research Institute, Princeton, NJ 08540, USA, joe@research.nj.nec.com

Abstract. This paper investigates one-round secure computation between two distrusting parties: Alice and Bob each have private inputs to a common function, but only Alice, acting as the receiver, is to learn the output; the protocol is limited to one message from Alice to Bob followed by one message from Bob to Alice. A model in which Bob may be computationally unbounded is investigated, which corresponds to information-theoretic security for Alice. It is shown that

1. for honest-but-curious behavior and unbounded Bob, any function computable by a polynomial-size circuit can be computed securely assuming the hardness of the decisional Diffie-Hellman problem;
2. for malicious behavior by both (bounded) parties, any function computable by a polynomial-size circuit can be computed securely, in a public-key framework, assuming the hardness of the decisional Diffie-Hellman problem.

The results are applied to secure autonomous mobile agents, which migrate between several distrusting hosts before returning to their originator. A scheme is presented for protecting the agent's secrets such that only the originator learns the output of the computation.

1 Introduction

Suppose Alice has a secret input x, Bob has a secret input y, and they wish to compute $g(x, y)$ securely using *one round* of interaction: Alice should learn $g(x, y)$ but nothing else about y and Bob should learn nothing at all. Communication is restricted to one message from Alice to Bob followed by one message from Bob to Alice. Without the restriction on the number of rounds, this is the problem of secure function evaluation introduced by Yao [26] and Goldreich et al. [17]. It is known that under cryptographic assumptions, every function can be computed securely and using a (small) constant number of rounds.

The problem is closely related to the question of "computing with encrypted data" [22]: Alice holds some input x, Bob holds a function f, and Alice should learn $f(x)$ in a one-round protocol, where Alice sends to Bob an "encryption" of x, Bob computes f on the "encrypted" data x and sends the result to Alice, who "decrypts" this to $f(x)$.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 512–523, 2000.
© Springer-Verlag Berlin Heidelberg 2000

The dual of this is "computing with encrypted functions," where Alice holds a function f, Bob holds an input y, and Alice should get $f(y)$ in a one-round protocol. This scenario has received considerable attention recently because it corresponds to protecting mobile code that is running on a potentially malicious host, which might be spying on the secrets of the code [24,25].

In the next paragraphs, honest-but-curious behavior is assumed before we turn to arbitrary malicious behavior. Honest-but-curious behavior models a passively cheating party who follows the protocol, but might try to infer illegitimate information later on.

Homomorphic Encryption and Computing with Encrypted Data. One popular approach to "computing with encrypted data" is to search for a public-key encryption scheme (E, D) with the following homomorphic property: given $E(x)$ and $E(y)$ one can efficiently compute $E(x + y)$ and $E(xy)$. Now, if Alice knows the private key D and sends Bob the public key E together with the encrypted data $E(x)$, then Bob can without interaction compute $E(f(x))$ and send it back to Alice. Although this has been a prominent open problem for years [15], it is still unknown whether such homomorphic encryption schemes exist. On the one hand, Boneh and Lipton [7] have shown that all such *deterministic* encryption schemes are insecure; on the other hand, Sander, Young, and Yung [25] propose a scheme that allows the necessary operations on encrypted data, but comes at the cost of a multiplicative blowup per gate, which limits the possible computations to functions with log-depth circuits.

Computational Assumptions. Note that the above approach to "computing with encrypted data" assumes a computationally bounded Bob, who cannot learn anything about the encrypted values. Alice, however, knows all secrets involved and seems not restricted in her computational power. Thus, the distinguishing feature of "computing with encrypted data" seems to be that it remains *secure against an unbounded Alice.* (In fact, the protocol of Sander et al. [25] is information-theoretically secure for Bob.)

Assume instead that Alice, the receiver of the output, is bounded and Bob is unbounded and consider the same question: is there a one-round secure computation protocol for all efficiently computable functions? We give a positive answer in Section 4.1: *any function computable by a polynomial-sized circuit has a one-round secure computation scheme in this model.* The result is obtained by combining Yao's "encrypted circuit" method [26] for secure computation with a one-round oblivious transfer protocol [4]. To our knowledge, this is the first one-round secure computation protocol for arbitrary polynomial-time computations and gives a partial answer to the long-standing open question of computing with encrypted data mentioned above.

If both parties are bounded, the above solution applies as well (we can even obtain stronger results, see below). Conversely, it is well known that secure computation between two unbounded parties with "full information" is impossible for arbitrary functions and limited to trivial functions g where $g(x, y)$ gives full

information about y. The following table summarizes the current state of one-round secure computation (both supply input, only Alice receives output):

Alice	Bob	securely computable functions	reference
unbounded	unbounded	only trivial ones	BGW [5]
unbounded	bounded	log-depth circuits	Sander et al. [25]
bounded	unbounded	polynomial-size circuits	this paper
bounded	bounded	polynomial-size circuits	this paper

Malicious Parties. We also investigate the *malicious model*, where both parties might be actively cheating. One cannot demand that Bob ever sends a second message, but if he does, and Alice accepts, the model ensures that Alice obtains $g(x, y)$ for her input x and some y. We show that if Alice and Bob are both computationally bounded, then a one-round protocol exists also in the malicious model, provided they share a random string and that Alice has a public key for which she is guaranteed to know the private key. This is a realistic model, which is also used elsewhere (e.g., [10]).

These results seem essentially optimal because one round of communication is needed to implement oblivious transfer [20].

Securing Autonomous Mobile Agents. One-round secure computation has been recognized as the solution for keeping the privacy of mobile code intact [24]. Here, a code originator O sends one message containing a protected description of the mobile code to host H, which "runs" the program and sends some output back to O, who decodes the output. (This is an instance of "computing with encrypted functions.") The results in this paper on one-round secure computation directly yield mobile code privacy for *all polynomial-time mobile computations.* This is a vast improvement over both the solutions of Sander and Tschudin [24] (which works for functions representable as polynomials) and the one of Sander et al. [25] (which works for functions computable by log-depth circuits).

In our solution the relative complexities of the computations by O and H are similar; for example, if H runs a long, complex computation with a short output, then O's decoding work is proportional to the complex computation, despite the output being short. We do not know if there are general schemes with "small" decoding complexity for O.

The above models are limited to mobile code that visits only one host, however. In Section 5, a protocol is presented that allows an autonomous mobile agent to visit *several distrusting hosts*, which need not be fixed ahead of time. This flexibility is one of the main benefits of the mobile code paradigm. As with an unencrypted autonomous agent, the communication flow must correspond to a closed path starting and ending at O. The secure computation protocol involves constructing a cascade of Yao-style circuits by the hosts and its evaluation by O. No host learns anything about the agent's or the other hosts' secrets.

Related Work. Protocols for two-party secure function evaluation between a bounded and an unbounded party have previously been proposed by Chaum, Damgård, and van de Graaf [11] and by Abadi and Feigenbaum [1]. The former

hides the inputs of one party information-theoretically and the latter hides the circuit information-theoretically from the other party (regardless of who receives the output). Both protocols have round complexity proportional to the depth of the circuit, however.

The work of Abadi, Feigenbaum, and Kilian [2] on information hiding from an oracle assumes an all-powerful oracle that helps a user with insufficient resources in computing $f(x)$ for his input x; the approach is to transform x into an encrypted instance y and have the oracle compute $f(y)$ such that it learns nothing about x but the user can infer $f(x)$ from $f(y)$. The two main differences to our model are (1) that Bob may also provide an input and (2) that the oracle is limited to computing $f(\cdot)$.

Feige, Kilian, and Naor [13] consider a related model in which two parties perform secure computation by sending a single message each to a third party.

2 Definitions

Recall the three scenarios of one-round secure computation introduced above: *computing with encrypted functions*, *computing with encrypted data*, and *secure function evaluation*. Using a universal circuit for g in secure function evaluation, it is straightforward to realize the first two scenarios from the third one by supplying f as input (at the cost of a polynomial expansion). An equivalence in the other directions is possible by letting f be g with one party's inputs fixed.

The remainder of this section presents definitions for one-round secure computation using secure function evaluation. Formal definitions may be constructed using the methods in [3,9,18] and are provided in the full version of the paper.

The security parameter is denoted by k and a quantity ϵ_k is called *negligible* (as a function of k) if for all $c > 0$ there exists a constant k_0 such that $\epsilon_k < \frac{1}{k^c}$ for all $k > k_0$. Throughout we assume that the security parameter k, as well as other system parameters, are always part of the input to all algorithms and protocols.

Honest-but-Curious Model. This definition captures one-round secure computation if both parties follow the protocol. A scheme has to ensure correctness, privacy for Alice, and privacy for Bob.

More precisely, a one-round secure computation scheme in the honest-but-curious model consists of three probabilistic polynomial-time algorithms $A_1(\cdot)$, $A_2(\cdot,\cdot)$, and $B(\cdot,\cdot)$ such that (1) $\forall x \in \mathcal{X}$, $\forall y \in \mathcal{Y}$, if $A_1(x)$ outputs (s, m_1) and $B(y, m_1)$ outputs m_2, then $A_2(s, m_2)$ outputs $g(x, y)$ with all but negligible probability; (2) there exists a simulator $\mathsf{sim}_{\mathsf{Bob}}$ that outputs (s, m_1) such that $\forall x \in \mathcal{X}$, no efficient algorithm can distinguish between the distributions output by $\mathsf{sim}_{\mathsf{Bob}}$ and the output of $A_1(x)$; (3) there exists a simulator $\mathsf{sim}_{\mathsf{Alice}}$ that outputs (s, m_1) such that $\forall x \in \mathcal{X}$ and $\forall y \in \mathcal{Y}$, if m_1 is computed from $A_1(x)$ and m_2 from $B(y, m_1)$, then no efficient algorithm can distinguish between the distributions on (x, m_1, m_2) induced by the real protocol and by $\mathsf{sim}_{\mathsf{Alice}}$.

We say that the scheme is secure for bounded Alice and *unbounded Bob* if the distinguisher in (2) is an arbitrary algorithm and the one in (3) is polynomial-

time; similarly, we say it is secure for *bounded Alice and bounded Bob* if both distinguishers are polynomial-time algorithms.

In the model above, A_1 is Alice's query generator that outputs a message m_1 sent to Bob and a secret s, B is Bob's algorithm that outputs message m_2 that is sent to Alice, and A_2 is Alice's decoding algorithm that interprets Bob's answer using s. (All algorithms are for a fixed function g.)

Malicious Model. The malicious model allows arbitrary behavior for (bounded) Alice and Bob. We must ensure that for every strategy of Alice, Bob's reply does not reveal more to her about y than what follows from the function output $g(x, y)$ on a particular x. Bob, on the other hand, must be bound to compute m_2 such that Alice can recover $g(x, y)$ for her x and on *some* legal y, chosen independently from x, or have Alice reject. Intuitively, this can be solved by having both parties supply a zero-knowledge proof with their message that it is well-formed. However, a formal proof of security requires that these proofs are proofs of knowledge. To this end, we use a public-key model [21], where each party has registered a public key and a public source of randomness is available (see Section 4.3).

3 Tools

3.1 Oblivious Transfer

A ubiquitous tool in secure computation is *oblivious transfer*. We use a one-out-of-two oblivious transfer also known as ANDOS (all-or-nothing-disclosure-of-secrets [8]): a sender S has two input values a_0 and a_1, which are strings of arbitrary known length, and a receiver R has a bit c; R obtains a_c, but should not learn anything about $a_{c\oplus 1}$ and S should not learn c.

Let G be a group of large prime order q (of length polynomial in k) such that $p = 2q + 1$ is prime and $G \subset \mathbb{Z}_p$ and let $g \in G$ be a generator. (Note that this allows efficient sampling from G with uniform distribution.) Consider two distributions D_0 and D_1 over G^4, where $D_0 = (g, g^\alpha, g^\beta, g^\gamma)$ with $g \xleftarrow{R} G$ and $\alpha, \beta, \gamma \xleftarrow{R} \mathbb{Z}_q$ and $D_1 = (g, g^\alpha, g^\beta, g^{\alpha\beta})$ with $g \xleftarrow{R} G$ and $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$. The *Decisional Diffie-Hellman (DDH) assumption* is that there exists no probabilistic polynomial-time algorithm that distinguishes with non-negligible probability between D_0 and D_1.

The following is a sketch of the ANDOS protocol between a sender Bob and a receiver Alice [4], denoted $\mathsf{OT}(c)(a_0, a_1)$. Alice's private input is a bit c and Bob's private inputs are $a_0, a_1 \in G$. Common inputs are p and g.

1. Bob chooses $\delta \xleftarrow{R} G$ and sends δ to Alice.
2. Alice chooses $\alpha \xleftarrow{R} \mathbb{Z}_q$, computes $\beta_c = g^\alpha$, $\beta_{c\oplus 1} = \delta/\beta_c$, and sends β_0, β_1 to Bob.
3. Bob verifies that $\beta_0\beta_1 = \delta$ and aborts if not. Otherwise, he chooses $r_0, r_1 \xleftarrow{R} \mathbb{Z}_q$, computes $(e_0, f_0) = (g^{r_0}, a_0{\beta_0}^{r_0})$ and $(e_1, f_1) = (g^{r_1}, a_1{\beta_1}^{r_1})$, and sends (e_0, f_0, e_1, f_1) to Alice.
4. Alice obtains a_c by computing $f_c/{e_c}^\alpha$.

It is easy to see that if both parties follow the protocol, Alice obtains a_c. Consider security for Alice: c is perfectly hidden from Bob, i.e., in an information-theoretic sense, because β_0 and β_1 are uniformly random among all group elements with product δ. Thus the protocol is secure for Alice against an arbitrarily behaving unbounded Bob.

Consider security for Bob. In the *honest-but-curious model*, Alice chooses β_0 and β_1 honest, i.e., such that $\beta_{c\oplus 1}$ is a random public key and, under the DDH assumption, $(e_{c\oplus 1}, f_{c\oplus 1})$ is a semantically secure encryption of $a_{c\oplus 1}$. Hence the protocol is secure for Bob against a bounded Alice. Furthermore, Step 1 of the protocol is not even needed and Alice may compute $\delta \xleftarrow{R} G$ herself in Step 2. The resulting protocol, denoted by $\mathsf{OT\text{-}1}(c)(a_0, a_1)$, has only one round of interaction.

Assuming *malicious behavior*, a one-round version is also possible in the public-key model using shared random information σ; this version is denoted by $\mathsf{OT\text{-}2}(c)(a_0, a_1)$. Here, Step 1 can again be omitted and Alice chooses δ herself, using the sampling algorithm in G with σ as random source. Intuitively, she then sends δ along with β_0, β_1 to Bob, who verifies that the choice of δ is correct according to σ. However, Alice must also supply a "non-interactive proof of knowledge" of α, the discrete logarithm of either β_0 or β_1 (we refer to Section 4.3 for how this can be done). With these changes, the protocol can be proved secure for Bob against an arbitrarily behaving bounded Alice.

3.2 Encrypted Circuit Construction

Yao's encrypted circuit construction implements secure function evaluation between Alice and Bob such that Alice receives the output $z = g(x, y)$ [26].

We give an abstract version of Yao's construction describing only those properties essential to our analysis. A more detailed treatment of Yao's protocol is found in the literature (e.g., [23]). Let $(x_1, \ldots, x_{n_x})$, $(y_1, \ldots, y_{n_y})$, and $(z_1, \ldots, z_{n_z})$ denote the binary representation of x, y, and z, respectively, and let C denote a polynomial-sized circuit computing $g(\cdot, \cdot)$. Yao's construction consists of three procedures: (1) an algorithm $\mathsf{construct}$ that Bob uses to construct an encrypted circuit, (2) an interactive protocol $\mathsf{transfer}$ between Alice and Bob, and (3) an algorithm $\mathsf{evaluate}$ allowing Alice to retrieve $g(x, y)$. Additionally, the proof of security requires a simulation result.

More precisely, the probabilistic algorithm $\mathsf{construct}(C, y)$ outputs the values $\mathcal{C}, (K_{1,0}, K_{1,1}), \ldots, (K_{n_x,0}, K_{n_x,1}), (U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1})$. The first part $\mathcal{C}$ is a representation for C, with input y hardwired in. It may be viewed as an encrypted version of the n_x-input circuit $C(\cdot, y)$. In order to compute $C(x, y)$, one needs a k-bit key for each input bit x_i; the key $K_{i,b}$ corresponds to the key used for the input $x_i = b$. The pairs $(U_{i,0}, U_{i,1})$ represent the output bits, i.e., if decryption of the circuit produces $U_{i,b}$, then the output bit z_i is set to b.

The $\mathsf{transfer}$ protocol consists of n_x parallel executions of ANDOS. In the i-th execution, Bob has input $(K_{i,0}, K_{i,1})$ and Alice has input x_i. That is, Alice learns $K_{1,x_1}, \ldots, K_{n_x,x_{n_x}}$, but nothing more, whereas Bob learns nothing about $x_1, \ldots, x_{n_x}$. Bob also sends $\mathcal{C}$ and $(U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1})$ to Alice.

The algorithm $\mathsf{evaluate}(\mathcal{C}, K_{1,x_1}, \ldots, K_{n_x,x_{n_x}})$ outputs either a special symbol `reject` or $U_{1,z_1}, \ldots, U_{n_z,z_{n_z}}$. From the latter Alice can recover z, and if Alice and Bob obey the protocol, then $z = g(x, y)$.

A key element of the security analysis is the existence of a polynomial-time simulator $\mathsf{sim}_{\mathsf{Yao}}(C, x, g(x, y))$ that outputs a tuple $\mathcal{C}, K_{1,x_1}, \ldots, K_{n_x,x_{n_x}}$, $(U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1})$; the distribution of the simulator's output is computationally indistinguishable from that induced on these same variables by $\mathsf{construct}(C, y)$ and x. Intuitively, given x and $g(x, y)$, the simulator can simulate Alice's view obtained by running Yao's protocol with an ideal (information-theoretically secure) ANDOS.

The existence of $\mathsf{construct}$, $\mathsf{evaluate}$, and $\mathsf{sim}_{\mathsf{Yao}}$ may be based on the existence of pseudo-random functions [16]; efficient implementations of pseudo-random functions can be based on the DDH assumption [19].

4 One-Round Secure Computation for Polynomial-Size Circuits

The basic idea of our one-round secure computation protocols is to combine the one-round oblivious transfer protocols with the encrypted circuit construction.

4.1 Honest Behavior

In the honest case, we use the one-round oblivious transfer protocol $\mathsf{OT\text{-}1}$ and send Bob's reply in $\mathsf{OT\text{-}1}$ along with the encrypted circuit computing g. The resulting scheme consists of the three following algorithms A_1, A_2, and B (using the notation above).

$A_1(x)$: Compute the first messages of Alice for n_x parallel oblivious transfer protocols: Let $(\delta^{(i)}, \beta_0^{(i)}, \beta_1^{(i)})$ be computed as in Step 2 of protocol $\mathsf{OT\text{-}1}$ with input x_i of Alice for $i = 1, \ldots, n_x$. Output $s = (\alpha^{(1)}, \ldots, \alpha^{(n_x)})$ and $m_1 = ((\delta^{(1)}, \beta_0^{(1)}, \beta_1^{(1)}), \ldots, (\delta^{(n_x)}, \beta_0^{(n_x)}, \beta_1^{(n_x)}))$.

$B(y, m_1)$: Invoke $\mathsf{construct}(C, y)$ to obtain $(\mathcal{C}, (K_{1,0}, K_{1,1}), \ldots, (K_{n_x,0}, K_{n_x,1}), (U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1}))$. Next, for each $i = 1, \ldots, n_x$, execute Step 3 of protocol $\mathsf{OT\text{-}1}$ using $(\delta^{(i)}, \beta_0^{(i)}, \beta_1^{(i)})$ (taken from m_1) and with Bob's inputs (a_0, a_1) set to $(K_{i,0}, K_{i,1})$. (Provided $|G|$ is sufficiently large, such an encoding of binary strings in G is possible.) Denote the output of this step by $m_{2,i} = (e_0^{(i)}, f_0^{(i)}, e_1^{(i)}, f_1^{(i)})$. Output $m_2 = (\mathcal{C}, m_{2,1}, \ldots, m_{2,n_x}, (U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1}))$.

$A_2(s, m_2)$: For $i = 1, \ldots, n_x$ execute Step 4 of protocol $\mathsf{OT\text{-}1}$ using x_i as c, $(e_{x_i}^{(i)}, f_{x_i}^{(i)})$ (taken from m_2) as (e_c, f_c), and $\alpha^{(i)}$ (taken from s) as α, hence recovering $K_{1,x_1}, \ldots, K_{n_x,x_{n_x}}$. Finally, invoke $\mathsf{evaluate}(\mathcal{C}, K_{1,x_1}, \ldots, K_{n_x,x_{n_x}})$ to obtain $U_{1,z_1}, \ldots, U_{n_z,z_{n_z}}$ and output z.

4.2 Analysis of the Honest-Party Case

Our description of Yao's protocol assumed an ideal implementation of ANDOS. We now analyze the above protocol using the oblivious transfer protocol $\mathsf{OT\text{-}1}$. When both parties are honest, the combined protocol's correctness follows easily from the correctness of Yao's protocol and the oblivious transfer protocol. To show privacy, we construct simulators for Alice's and Bob's views. Note that this separation of privacy and correctness is not valid for parties with arbitrary behavior.

Let $View_{Alice}(x, y)$ and $View_{Bob}(x, y)$ denote Alice's and Bob's view of the protocol (C is always a fixed common input, and is dropped for notational convenience). We must simulate each player's view given only the information they are supposed to learn. That is, Bob is allowed to learn y, and Alice is allowed to learn x and $g(x, y)$.

To simulate $View_{Bob}(x, y)$, we define simulator $\mathsf{sim}_{\mathsf{Bob}}(y)$ as follows:

1. Choose Alice's input $x = 0^{n_x}$.
2. Engage in the secure function evaluation protocol with Bob where the simulated Alice plays as she would be given x. Return the view obtained by Bob during the execution of this protocol.

Lemma 1. *For all values of* (C, y), $\mathsf{sim}_{\mathsf{Bob}}(y)$ *and* $View_{Bob}(y)$ *are identically distributed.*

The proof follows from the fact that in every execution of the $\mathsf{OT\text{-}1}$ sub-protocol, Alice's message is independent of her input.

Next, we simulate $View_{Alice}(x, y)$. Define $\mathsf{sim}_{\mathsf{Alice}}(x, g(x, y))$ as follows:

1. The simulator invokes the simulator $\mathsf{sim}_{\mathsf{Yao}}(C, x, g(x, y))$ so as to obtain $\mathcal{C}, K_{1,x_1}, \ldots, K_{n_x,x_{n_x}}, (U_{1,0}, U_{1,1}), \ldots, (U_{n_z,0}, U_{n_z,1})$.
2. For $i = 1, \ldots, n_x$, the simulator chooses $K_{i,x_i \oplus 1} = 0^k$.
3. The simulator engages in the protocol $\mathsf{transfer}$ with Alice exactly as would Bob, given input pairs $(K_{1,0}, K_{1,1}), \ldots, (K_{n_x,0}, K_{n_x,1})$ and encrypted circuit $\mathcal{C}$. The simulator returns Alice's view of this protocol.

Lemma 2. *For all values of* x *and* y, $View_{Alice}(x, y)$ *and* $\mathsf{sim}_{\mathsf{Alice}}(x, g(x, y))$ *are computationally indistinguishable.*

The proof works by a hybrid argument (omitted). Our first result follows.

Theorem 1. *Under the DDH assumption,* (A_1, A_2, B) *are a one-round secure computation scheme in the honest-but-curious model, with perfect security against unbounded Bob.*

4.3 Allowing Malicious Behavior

For polynomially bounded, arbitrarily malicious parties, we obtain secure one-round computation in a model with certified public-keys and public randomness.

First, because we can no longer trust Alice to choose δ at random, we replace protocol OT-1 by protocol OT-2 (using public randomness) in the above construction. Then Bob must prove that his messages in OT-2 are consistent with a correct construction of $\mathcal{C}, \{(K_{i,0}, K_{i,1})\}$ and Alice must prove that she knows the discrete logarithm for one element of each pair $(\beta_0^{(i)}, \beta_1^{(i)})$ (e.g., using a result by Cramer et al. [12]). In the security proof for protocol OT-2 one extracts the discrete logarithms from Alice and thereby obtains her input x (x_i corresponds to the element of $(\beta_0^{(i)}, \beta_1^{(i)})$ of which Alice knows the discrete logarithm).

A *fallacious* step would be to use a public random string to implement non-interactive zero-knowledge proofs (NIZKP) [6,14] that each player's message is well formed. The formal complication to this method is that "standard" NIZKP are *not* proofs of knowledge. Instead, we use the "public-key" scenario for non-interactive proofs of knowledge, put forth by Simon and Rackoff [21], as follows. Each player has a public-key P that is certified by some trusted center once and forever. The player convinces the center, via a standard zero-knowledge proof of knowledge, that he knows the corresponding secret key S for P. Henceforth, the secret key is assumed available to the simulator/extractor. To make a non-interactive proof of knowledge of the solution to some problem in *NP*, the player simply encrypts, using P, whatever it is he wishes to show knowledge of, and then non-interactively prove (using standard NIZKP) that the encryption, if decrypted, would yield a solution to the problem. The extractor, who knows S, can then recover the solution as well. Details are omitted from this extended abstract.

5 Securing Autonomous Mobile Agents

The mobile agent paradigm has several attractive features. One of them is the flexibility of delegating a task to an autonomous agent, who roams the net, visits different sites, collects information, computes intermediate results, and returns to the originator only when the computation is finished. No interaction with the originator is needed in-between.

Sander and Tschudin [24] recognized that mobile code can be protected against a curious host using the approach of "computing with encrypted functions." However, their solution addresses only the case of agents who return home after visiting one host. We consider *autonomous agents* here that leave the originator without a fixed list of hosts to visit in mind and consider the question: How does the agent migrate securely from one host to another?

5.1 Model

More formally, there is the agent's originator O and ℓ hosts $H_1, \ldots, H_\ell$ that run the agent. The state of the agent is represented by some $x \in \mathcal{X}$. The initial state is chosen by O. All that is known about the computation is represented by $g_j : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}$ associated with H_j, which updates the agent's state according

to H_j's secret input y_j. This models arbitrary polynomial-time computations provided the functions g_j are representable by polynomial-size circuits.

A novel feature of our protocol is that neither the hosts nor the path taken by the agent need to be fixed or known beforehand. Only for the simplicity of description do we assume that the agent travels from O to host H_1, then from host H_j to host H_{j+1} for $j = 1, \ldots, \ell - 1$ and then from H_ℓ back to O; the generalization can be derived easily. The agent is autonomous because either a host may decide where to send the agent next or because the agent, besides the encrypted part, consists also of conventional code that computes where to go next based on non-private information. (Note that migration decisions cannot depend on the private state of the computation, as they would be observable by the hosts and thereby leak information about the internal state!)

A scheme for *secure computation by autonomous mobile agents* consists of efficient algorithms $A_1(\cdot)$, $A_2(\cdot,\cdot)$, $B_1(\cdot,\cdot)$, ... , $B_\ell(\cdot,\cdot)$. The agent's computation proceeds as follows: first, O runs $A_1(x)$ on input x and thereby obtains a secret s and a message m_0, which O sends to H_1; likewise, for $j = 1, \ldots, \ell$, H_j runs $B_j(y_j, m_{j-1})$ on input y_j, message m_{j-1} and obtains m_j, which it sends to H_{j+1} (with the exception that H_ℓ sends m_ℓ to O). Finally, upon receiving m_ℓ, O obtains the desired result by invoking $A_2(s, m_\ell)$. We require:

Correctness: $\forall x \in \mathcal{X}$ and $\forall y_j \in \mathcal{Y}$, the decoding algorithm $A_2(s, m_\ell)$ outputs $z = g_\ell(\cdots g_2(g_1(x, y_1), y_2) \cdots, y_\ell)$;

Privacy: (1) the inputs and computations of the visited hosts remain hidden from the other hosts: for all j, message m_j does not give information about x and $y_{j'}$ for $j' \leq j$; (2) the originator should learn only the output of the computation, but nothing else about the inputs of the hosts: $\forall x \in \mathcal{X}$ and $\forall y_j \in \mathcal{Y}$, $(j = 1, \ldots, \ell)$, given only x, s, and z (as above), m_ℓ can be simulated efficiently.

Honest-but-curious behavior is assumed on behalf of all parties throughout this section (dishonest behavior can be prevented analogously to the two-party case).

5.2 Protocol

Our protocol for secure computation by autonomous mobile agents is an extension of the one-round secure computation protocol in Section 4.1 to multiple hosts, which take over the part of Bob. O proceeds as Alice, sending the first message and receiving the encrypted circuit computing $g_\ell(\cdots(g_1(x, y_1), y_2) \cdots, y_\ell)$. Each host H_j contributes the part of encrypted circuit representing its function g_j; thus the resulting encrypted circuit is a cascade of sub-circuits. H_1 generates the key pairs representing O's input and computes the answers for the oblivious transfer protocol; these are attached to the computation and reach O with the message from H_ℓ. To extend the cascade of sub-circuits, H_j encrypts each input key of its sub-circuit with the corresponding output key from the preceding sub-circuit. This is done using a symmetric encryption algorithm $\mathsf{enc}_K(\cdot)$, realized in the same way as the encryptions for single gates in Yao's construction; in

particular, this scheme has the property that given a key K, one can efficiently check if a ciphertext represents an encryption under key K.

We describe algorithms A_1, A_2, and $B_1, \ldots, B_\ell$ using notation from above.

$A_1(x)$: Compute the first message (of Alice) for n_x parallel oblivious transfer protocols. This results in $s = (\alpha^{(1)}, \ldots, \alpha^{(n_x)})$ and $\widetilde{m}_0 = \big((\delta^{(1)}, \beta_0^{(1)}, \beta_1^{(1)}), \ldots, (\delta^{(n_x)}, \beta_0^{(n_x)}, \beta_1^{(n_x)})\big)$ computed as in $\mathsf{OT\text{-}1}$. Output s and $m_0 = (\widetilde{m}_0, \emptyset)$.

$B_j(y_j, m_{j-1})$: Invoke $\mathsf{construct}(C_j, y_j)$ to obtain

$$\mathcal{C}_j, (K_{1,0}^{(j)}, K_{1,1}^{(j)}), \ldots, (K_{n_x,0}^{(j)}, K_{n_x,1}^{(j)}), (U_{1,0}^{(j)}, U_{1,1}^{(j)}), \ldots, (U_{n_x,0}^{(j)}, U_{n_x,1}^{(j)}).$$

If $j = 1$, then execute Step 3 of protocol $\mathsf{OT\text{-}1}$ using $(\delta^{(i)}, \beta_0^{(i)}, \beta_1^{(i)})$ (taken from $\widetilde{m}_0$) and with Bob's input set to $(K_{i,0}^{(1)}, K_{i,1}^{(1)})$. Denote the output of the $\mathsf{OT\text{-}1}$ step by $\widetilde{m}^{(i)} = (e_0^{(i)}, f_0^{(i)}, e_1^{(i)}, f_1^{(i)})$. Set $\widetilde{m}_1 = (\widetilde{m}^{(1)}, \ldots, \widetilde{m}^{(n_x)}, \mathcal{C}_1)$ and output $m_1 = (\widetilde{m}_1, (U_{1,0}^{(1)}, U_{1,1}^{(1)}), \ldots, (U_{n_x,0}^{(1)}, U_{n_x,1}^{(1)}))$.
If $1 < j \leq \ell$, then the outputs of $\mathcal{C}_{j-1}$ are recoded as inputs to $\mathcal{C}_j$. To this end, for $i = 1, \ldots, n_x$ do the following: choose a random bit c_i and, for $b \in \{0, 1\}$, encrypt key $K_{i,b}^{(j)}$ under $U_{i,b}^{(j-1)}$ (taken from m_{j-1}) as $V_{i,b\oplus c_i}^{(j)} = \mathsf{enc}_{U_{i,b}^{(j-1)}}(K_{i,b}^{(j)})$ Next, set $\widetilde{m}_j = (\widetilde{m}_{j-1}, \mathcal{C}_j, (V_{1,0}^{(j)}, V_{1,1}^{(j)}), \ldots, (V_{n_x,0}^{(j)}, V_{n_x,1}^{(j)}))$ and then output $m_j = (\widetilde{m}_j, (U_{1,0}^{(j)}, U_{1,1}^{(j)}), \ldots, (U_{n_x,0}^{(j)}, U_{n_x,1}^{(j)}))$.

$A_2(s, m_\ell)$: Run Step 4 of protocol $\mathsf{OT\text{-}1}$ and obtain input keys $K_{1,x_1}^{(1)}, \ldots, K_{n_x,x_{n_x}}^{(1)}$ of $\mathcal{C}_1$. Now, run algorithm $\mathsf{evaluate}(\mathcal{C}_1, K_{1,x_1}^{(1)}, \ldots, K_{n_x,x_{n_x}}^{(1)})$ to obtain the output keys of $\mathcal{C}_1$. Each one of these decrypts one ciphertext $V_{i,b}^{(2)}$ to an input key of $\mathcal{C}_2$, which can then be evaluated and then will allow to decrypt the input keys of $\mathcal{C}_3$. Proceeding similarly for all circuits $\mathcal{C}_3, \ldots, \mathcal{C}_\ell$ will eventually reveal $U_{1,z_1}^{(\ell)}, \ldots, U_{n_x,z_{n_x}}^{(\ell)}$ from which the result z can be retrieved.

As for the security of the protocol, note that each host sees an encrypted circuit representing the computation so far, like Alice in the original protocol but lacking the secrets to decrypt the oblivious transfers. A simulator for each host's view is straightforward. When the encrypted circuit reaches O, it consists only of information that has been constructed using the same method as in the original protocol; thus, the security follows from the original argument.

References

1. M. Abadi and J. Feigenbaum, "Secure circuit evaluation: A protocol based on hiding information from an oracle," *Journal of Cryptology*, vol. 2, pp. 1–12, 1990.
2. M. Abadi, J. Feigenbaum, and J. Kilian, "On hiding information from an oracle," *Journal of Computer and System Sciences*, vol. 39, pp. 21–50, 1989.
3. D. Beaver, "Foundations of secure interactive computing," in *Proc. CRYPTO '91* (J. Feigenbaum, ed.), LNCS 576, 1992.
4. M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications," in *Proc. CRYPTO '89* (G. Brassard, ed.), LNCS 435, pp. 547–557, 1990.

5. M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proc. 20th STOC*, pp. 1–10, 1988.
6. M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge proof systems and its applications," in *Proc. 20th STOC*, pp. 103–112, 1988.
7. D. Boneh and R. J. Lipton, "Searching for elements in black box fields and applications," in *Proc. CRYPTO '96*, LNCS 1109, 1996.
8. G. Brassard, C. Crépeau, and J.-M. Robert, "Information theoretic reductions among disclosure problems," in *Proc. 27th FOCS*, 1986.
9. R. Canetti, "Security and composition of multi-party cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.
10. R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali, "Resettable zero-knowledge," in *Proc. 32nd STOC*, 2000.
11. D. Chaum, I. Damgård, and J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result," in *Proc. CRYPTO '87* (C. Pomerance, ed.), LNCS 293, 1988.
12. R. Cramer, I. Damgård, and B. Schoemakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Proc. CRYPTO '94* (Y. G. Desmedt, ed.), LNCS 839, 1994.
13. U. Feige, J. Kilian, and M. Naor, "A minimal model for secure computation (extended abstract)," in *Proc. 26th STOC*, pp. 554–563, 1994.
14. U. Feige, D. Lapidot, and A. Shamir, "Multiple noninteractive zero knowledge proofs under general assumptions," *SIAM Journal on Computing*, vol. 29, no. 1, pp. 1–28, 1999.
15. J. Feigenbaum and M. Merritt, "Open questions, talk abstracts, and summary of discussions," in *Distributed Computing and Cryptography*, AMS, 1991.
16. O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM*, vol. 33, pp. 792–807, Oct. 1986.
17. O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *Proc. 19th STOC*, pp. 218–229, 1987.
18. S. Micali and P. Rogaway, "Secure computation," in *Proc. CRYPTO '91* (J. Feigenbaum, ed.), LNCS 576, pp. 392–404, 1992.
19. M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudorandom functions," in *Proc. 38th FOCS*, 1997.
20. R. Ostrovsky, R. Venkatesan, and M. Yung, "Fair games against an all-powerful adversary," in *Advances in Computational Complexity Theory*, AMS, 1993.
21. C. Rackoff and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," in *Proc. CRYPTO '91* (J. Feigenbaum, ed.), LNCS 576, pp. 433–444, 1992.
22. R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of Secure Computation* (R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, eds.), pp. 169–177, Academic Press, 1978.
23. P. Rogaway, *The Round Complexity of Secure Protocols.* PhD thesis, MIT, 1991.
24. T. Sander and C. F. Tschudin, "Protecting mobile agents against malicious hosts," in *Mobile Agents and Security* (G. Vigna, ed.), LNCS 1419, 1998.
25. T. Sander, A. Young, and M. Yung, "Non-interactive CryptoComputing for NC^1," in *Proc. 40th FOCS*, 1999.
26. A. C. Yao, "How to generate and exchange secrets," in *Proc. 27th FOCS*, pp. 162–167, 1986.

Round-Optimal and Abuse-Free Optimistic Multi-party Contract Signing

Birgit Baum-Waidner[1] and Michael Waidner[2]

[1] Entrust Technologies (Switzerland), birgit.baum@entrust.com
[2] IBM Zurich Research Laboratory, wmi@zurich.ibm.com

Abstract. We present the first optimistic n-party contract signing protocol for asynchronous networks that tolerates up to $n-1$ dishonest signatories and terminates in the minimum number of rounds ($O(n)$). We also show how to make this protocol abuse-free by using standard cryptographic primitives (digital signatures, public-key encryption) only. Previous solutions required $O(n^2)$ rounds of communication, and non-standard cryptographic primitives for abuse freeness.

1 Introduction

A *contract signing protocol* is a protocol that allows n signatories to sign a contract text such that, even if up to $n-1$ of them are dishonest, either *all* honest signatories obtain a signed contract, or nobody obtains it [2,8].

Dishonest signatories can arbitrarily deviate from their protocol (Byzantine model). We assume an *asynchronous* network: all messages are eventually delivered, but there are no upper bounds on network delays.[1]

Multi-party contract signing has obvious applications in secure electronic commerce, and is the basis for the solution of many related fairness problems, like multi-party certified mail [1].

By using a *third party,* T, that is always honest, the problem can be trivially solved [18]: T collects a digital signature from each of the n signatories, and either redistributes the n signatures, or aborts the contract in case not all n signatures arrive. Security depends fully on T; therefore most research has been focused on getting rid of T as trust and performance bottleneck.

Unfortunately, one cannot get rid of T completely: For $n = 2$ no deterministic protocol without third party exists [11], and each probabilistic protocol has an error probability at least linear in the number of rounds [7]. These results apply also to the n-party case, provided a majority of all signatories might be dishonest.

Therefore our goal is to minimize the involvement of T as much as possible: *Optimistic* protocols depend on a third party T, but in such a way that T is *not* actively involved in case all signatories are honest; only for recovery purposes T might become active [3,7,16].

Optimistic contract signing protocols have been first described for *synchronous* networks [2,3,16,17]. 2-party protocols for *asynchronous* networks have

[1] Actually we do not need to require eventual delivery for all messages; see Sect. 2.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 524–535, 2000.
© Springer-Verlag Berlin Heidelberg 2000

been described in [4,12,17]. The n-party case was first investigated in [5]. Independently, but subsequently, protocols for the 3-party and n-party case were proposed in [12] and [13], resp. In Sect. 3 we give a precise definition of the asynchronous n-party contract signing problem.

In Sect. 4 we describe and prove our new asynchronous optimistic n-party contract signing protocol. The protocol is optimistic in case all signatories agree to sign the contract. It requires $t + 4$ rounds of communication in the worst case, $t + 2$ rounds in the optimistic case, and $O(tn^2)$ messages, where $t < n$ is a protocol parameter describing the maximum number of dishonest signatories. A variant requires $2t + 6$ rounds but only $O(tn)$ messages. For $t = n - 1$ any such protocol requires $O(n)$ rounds [13], thus, our protocol is asymptotically *round-optimal.* No previous protocol achieved this bound; the protocol proposed in [13] requires $O(n^2)$ rounds and $O(n^3)$ messages.

In Sect. 5 we add *abuse freeness* to the protocol of Sect. 4: A protocol can be *abused* if at some point in time the dishonest signatories fully control the result (i.e., depending on how they behave the result will be *signed* or *failed*, irrespective of the honest signatories' behavior) and can *prove* to an outside party that they can force the result to be *signed* [12].

Two-party abuse-free protocols have been introduced in [12] and, although without mentioning this feature, in [4]. The 3-party and n-party cases were considered in [12] and [13], respectively.

Our construction for abuse freeness is significantly simpler than the construction in [12,13]: it uses standard cryptographic primitives only (public-key encryption, digital signatures), i.e., unlike [12,13] we do not need specific primitives. The construction preserves the round optimality.

2 Model and Notation

Parties. Let $P_1, \ldots, P_n$ denote the *signatories,* T a *third party,* and $V_1, \ldots, V_{n'}$ the potential *verifiers* of signed contracts. Each *party* represents a machine that executes a certain protocol and that servers a specific user (e.g., human being or another protocol). Parties can receive *inputs* from their users (e.g., a signatory the command to sign a certain contract) and can generate certain *outputs* for their users (e.g., the result of the protocol).

The signatories P_i and third party T are able to *digitally sign* messages, and all parties are able to verify their signatures [14]. The signature on message m associated with P_X (T) is denoted by $\mathsf{sign}_X(m)$ ($\mathsf{sign}_T(m)$). In our complexity analyses we assume that $\mathsf{sign}(m)$ has constant length, independent of the length of m (e.g., because m is hashed before signing [15]). We abstract from the error probabilities introduced by cryptographic signature schemes and assume that signatures are unforgeable (the translation into a cryptographic model is straightforward).

Adversary model. We assume that up to t (for a given $t < n$) of the n signatories, and for some requirements also T and all verifiers might be *dishonest.*

Dishonest parties can behave arbitrarily and are coordinated by a single party, called the *adversary.* Security properties must always hold for *all* adversaries.

Network. All messages sent to or from T or any V_i are reliably delivered, eventually. We do not require any particular level of synchronization, nor do we assume that messages are delivered in order. The decision on which of all sent messages to deliver next is taken by the adversary. The adversary can read all messages from, and can insert additional messages into all channels.

All-honest case. This is a special case where we assume that *all* n signatories are honest and *all* messages sent are reliably delivered, eventually. In Sect. 5 we assume, additionally, that in this case, the adversary does not insert any messages into the channels between signatories.[2]

We make use of a few conventions, in order to simplify presentation:

Timeouts. If we say that *"party X waits for a certain set of messages, M, but can stop waiting any time"* we mean more precisely the following: X accepts a special input wakeup from its user, for each protocol execution. "Waiting for M" means that X continues to receive messages but does not proceed in the protocol run until either all messages in M have been received, or wakeup is input. (In case wakeup is input *before* X starts waiting for M it proceeds immediately.)

Initial inputs and protocol parameters. We assume that the parties and their public keys are known and fixed in advance.

3 Definitions

Definition 1 (Asynchronous Multi-party Contract Signing). An *Asynchronous Multi-Party Contract Signing Scheme* (asynchronous MPCS) consists of two protocols:

- $\mathsf{sign}[P_1, \ldots, P_n]$, for signing a contract with signatories $P_1, \ldots, P_n$. sign[] might involve an additional party, T, in which case we call it an *MPCS with third party.*
- $\mathsf{verify}[P_i, V_j]$, for showing a signed contract to any verifier, V_j, $j \in \{1, \ldots, n'\}$. verify[] never involves T, i.e., it is always a 2-party protocol only.

A signatory P_i starts sign[] on input $(\mathsf{sign}, tid_i, contr_i)$.[3] tid_i is a transaction identifier unique for all executions of sign[]. $contr_i$ is the contract text to be signed.

Upon termination sign[] produces an output $(tid_i, contr_i, d_i)$ for P_i, with $d_i \in \{\mathsf{signed}, \mathsf{failed}\}$. We will simply say "$P_i$ decides d_i." P_i may receive an input (wakeup, tid_i) any time.

[2] This assumption is not needed for a weaker version of abuse freeness; see [6].

[3] If the user of P_i does not wish to sign then P_i does not participate in sign[]. One could also allow an additional input $decision_i \in \{\mathsf{sign}, \mathsf{reject}\}$, indicating whether the user wants to sign or not, and assume that in the all-honest case each signatory receives an input and participates in sign[]. This would enable faster termination in case one signatory gets input reject.

A signatory P_i starts verify[] with verifier V_j on input $(\mathsf{show}, V_j, tid_i, contr_i)$. A verifier V_j starts verify[] on input $(\mathsf{verify}, P_i, tid_V, contr_V)$. Upon termination $\mathsf{verify}[P_i, V_j]$ produces an output $(tid_V, contr_V, d_V)$ with $d_V \in \{\mathsf{signed}, \mathsf{verify_failed}\}$ for V_j. We will simply say "V_j decides d_V." No output is produced for P_i. V_j may receive an input (wakeup, tid_V) any time.

The following requirements must be satisfied:

(R1) *Correct execution.* In the all-honest case, if all signatories start with the same input (sign, *tid*, *contr*), and no signatory receives (wakeup, *tid*), then all signatories terminate and decide signed.

(R2) *Unforgeability.* If an honest P_i never received input $(\mathsf{sign}, tid, contr)$ then *no* honest V_j that receives input $(\mathsf{verify}, P_{i'}, tid, contr)$, for any $P_{i'}$, will decide signed. (Note that this does not assume an honest T.)

(R3) *Verifiability of valid contracts.* If an honest P_i decides signed on input (sign, *tid*, *contr*), and later P_i receives input $(\mathsf{show}, V_j, tid, contr)$ and honest V_j receives input $(\mathsf{verify}, P_i, tid, contr)$ and does not receive (wakeup, *tid*) afterwards then V_j will decide signed.

(R4) *No surprises with invalid contracts.* If T is honest, and an honest P_i received input $(\mathsf{sign}, tid, contr)$ but decided failed then no honest verifier V_j receiving $(\mathsf{verify}, P_{i'}, tid, contr)$, for any $P_{i'}$, will decide signed.

(R5) *Termination of* sign[]. If T is honest then each honest P_i that receives sign and wakeup (for the same *tid* and *contr*) will terminate eventually.

(R6) *Termination of* verify[]. Each honest V_j that receives verify and then wakeup, for the same *tid*, and each honest P_i that receives show will terminate eventually.

Definition 2 (Optimistic Protocol). An MPCS with third party T is called *optimistic on agreement* if in the all-honest case, if all signatories receive input (sign, *tid*, *contr*) and none receives (wakeup, *tid*) the protocol terminates without T ever sending or receiving any messages.

It is called *optimistic on disagreement* if in the all-honest case, if some signatories do *not* receive input (sign, *tid*, *contr*) the protocol terminates without T ever sending or receiving any messages.

It is called *optimistic* if it is optimistic on agreement and on disagreement.

Definition 3 (Abuse freeness [12]). An MPCS is called *abuse-free* if it is impossible for the adversary at any point in the protocol to be able to prove to an outside party that he has the power to control the contract, i.e., to terminate the protocol with result signed or failed.

Remark 1. Def. 1 captures *plain* contract signing only. We do not require additional features, like fair exchange of pre-defined signatures [4] or invisibility of T [16]. If required, they can be implemented on top of contract signing.

4 Asynchronous Optimistic Multi-party Contract Signing

The following Scheme 1 solves the multi-party contract signing problem, and is optimistic on agreement.

Ignoring all details, the protocol works as follows: It consists of $t+2$ locally defined rounds. In Round 1 each signatory that starts the protocol signs a "promise" to sign the contract and broadcasts this promise. In each subsequent round each signatory collects all signatures from the previous round, countersigns this set of n signatures, and broadcasts it.[4] The result of the $(t+2)$-nd round becomes the real contract.

A signatory who becomes tired of waiting for some signatures in some round sends the information received so far to the honest T, stops sending further messages and simply waits for T's answer. T analyses the situation and decides either failed or signed:

If the first request received by T comes from a signatory in the *first* round then T must decide failed. If T receives a request from the *last* round then T must decide signed, as other signatories might already have the signed contract. Thus if T receives requests from all rounds then somewhere in the middle between the first and the last round T might have to change the decision from failed to signed. But T must do this only if it is clear that all requests that were answered with failed came from dishonest signatories—otherwise two honest signatories might decide differently.

Scheme 1 (Asynchronous Optimistic Multi-party Contract Signing)

Protocol "sign" for honest P_i:

The protocol is given by the following rules; P_i applies them, sequentially, until it *stops*. Let $c_i := (tid_i, contr_i)$.

The protocol will proceed in locally defined rounds. Let $r := 1$ a local round counter for P_i, and let *raised_exception* := false a Boolean variable indicating whether P_i contacted T or not. Both are initialized before executing any rule. Let $M_{0,i} :=$ nil, for all i.

- **Rule S1:** If *raised_exception* = false and $r = 1$ then:
  - P_i sends $m_{1,i} := \mathsf{sign}_i(c_i, 1, \mathsf{prev_rnd_ok})$ to all signatories.
  - From all received messages of type $m_{1,j}$ it tries to compile full and consistent vectors $M_{1,i} := (m_{1,1}, \ldots, m_{1,n})$ and $X_{1,i} := M_{1,i}$ with $m_{1,j} = \mathsf{sign}_j(c_i, 1, \mathsf{prev_rnd_ok})$. (NB: here we also check that all signatories agree on c_i.) If this succeeds P_i sets $r := 2$.

  At any time P_i can stop waiting for any missing $m_{1,j}$ (i.e., the user of P_i might enter wakeup any time), in which case it sets *raised_exception* := true and sends $resolve_{1,i} := (1, i, \mathsf{sign}_i(m_{1,i}, \mathsf{resolve}))$ to T.
- **Rule S2:** If *raised_exception* = false and $2 \le r \le t+2$ then:
  - P_i sends $m_{r,i} := (\mathsf{sign}_i(M_{r-1,i}, r, \mathsf{vec_ok}), \mathsf{sign}_i(c_i, r, \mathsf{prev_rnd_ok}))$ to all signatories.
  - From all received messages of type $m_{r,j}$ it tries to compile full and consistent vectors

$$M_{r,i} = (\mathsf{sign}_1(c_i, r, \mathsf{prev_rnd_ok}), \ldots, \mathsf{sign}_n(c_i, r, \mathsf{prev_rnd_ok}))$$

[4] The real protocol does this more efficiently, in order to keep the messages short.

$X_{r,i} := (\mathsf{sign}_1(M_{r-1,i}, r, \mathsf{vec_ok}), \ldots, \mathsf{sign}_n(M_{r-1,i}, r, \mathsf{vec_ok}))$

If this succeeds and

$r < t+2$ then it sets $r := r+1$;

$r = t+2$ then it decides signed, sets $C_i := M_{r,i}$, and *stops.*

At any time P_i can stop waiting for any missing $m_{r,j}$. In this case P_i sets *raised_exception* := true and sends $resolve_{r,i} := (r, i, \mathsf{sign}_i(X_{r-1,i}, \mathsf{resolve}), X_{r-1,i}, M_{r-2,i})$ to T.

– **Rule S3:** If *raised_exception* = true then:
P_i waits for a message from T (without sending or receiving any further messages). This can be any of $signed_{r,i} = (resolve_{r',j}, \mathsf{sign}_T(c, r', j, \mathsf{signed}))$ or $aborted_{r,i} = \mathsf{sign}_T(c_i, r, i, \mathsf{aborted})$. On receiving one P_i decides as indicated by T and *stops*; if it decides signed it sets $C_i := signed_{r,i}$.

Protocol "sign" for third party T:

We assume T receives a message $resolve_{r,i}$, as defined in the protocol for P_i. Let c be the (unique) pair (*tid*, *contr*) contained in $resolve_{r,i}$.

If this is the first time T is asked about this contract (i.e., *tid*) then T initializes a Boolean variable *signed* := false and two sets *con* := $\emptyset$ and *abort_set* := $\emptyset$. *signed* indicates T's current decision, signed or failed. *con* is the set of all indices of signatories that contacted T, and *abort_set* is the set of all failed-messages sent by T.

Processing $resolve_{r,i}$ cannot be interrupted, i.e., it cannot happen that T processes two different $resolve_{r,i}$ concurrently.

– **Rule T0:** *(T accepts only one request from each P_i. All other requests are ignored.)* If $i \in con$ then the message $resolve_{r,i}$ is ignored.
– **Rule T1:** *(If T receives a request from Round $r = 1$ and has not yet decided* signed *then T sends an* abort *to the requester.)* If $i \notin con$ and *signed* = false and $r = 1$ then T sets

$aborted_{r,i} := \mathsf{sign}_T(c, r, i, \mathsf{aborted})$
$abort_set := abort_set \cup \{aborted_{r,i}\}$
$con := con \cup \{i\}$

and sends $aborted_{r,i}$ to P_i.
– **Rule T2:** *(If T receives a request from a Round $r > 1$ and has not yet decided* signed *then T checks whether all previous requests came from dishonest signatories, using Lemma 1. If this is the case then it changes the decision to* signed*, otherwise T sticks to* aborted*.)*
If $i \notin con$, *signed* = false, $r > 1$, and
if for all $aborted_{s,k} \in abort_set$ we have $s < r-1$

- then: If $con = \emptyset$ then T sets *first_signed* := $(resolve_{r,i}, \mathsf{sign}_T(c, r, i, \mathsf{signed}))$. T sets

  $signed_{r,i} := first_signed$
  $signed := \mathsf{true}$
  $con := con \cup \{i\}$

  and sends $signed_{r,i}$ to P_i.

- else: T sets
  $aborted_{r,i} := \mathsf{sign}_T(c, r, i, \mathsf{aborted})$
  $abort_set := abort_set \cup \{aborted_{r,i}\}$
  $con := con \cup \{i\}$
  and sends $aborted_{r,i}$ to P_i.

– **Rule T3:** *(A decision* signed *is always preserved.)* If $i \notin con$ and $signed = \mathsf{true}$ then T sets
  $signed_{r,i} := first_signed$
  $con := con \cup \{i\}$
  and sends $signed_{r,i}$ to P_i.

Protocol "verify":

If P_i wants to show a signed contract on c to verifier V it sends C_i to V. V outputs signed if it receives a messages C_i from P_i such that

– **Rule V1:** $C_i = (\mathsf{sign}_1(c, t+2, \mathsf{prev_rnd_ok}), \ldots, \mathsf{sign}_n(c, t+2, \mathsf{prev_rnd_ok}))$, or
– **Rule V2:** $C_i = ((2, j, \mathsf{sign}_j(X_{1,j}, \mathsf{resolve}), X_{1,j}, M_{0,j}), \mathsf{sign}_T(c, 2, j, \mathsf{signed}))$, for some j, $X_{1,j} = (\mathsf{sign}_1(c, 1, \mathsf{prev_rnd_ok}), \ldots, \mathsf{sign}_n(c, 1, \mathsf{prev_rnd_ok}))$ and $M_{0,j} = \mathsf{nil}$, or
– **Rule V3:** $C_i = ((r, j, \mathsf{sign}_j(X_{r-1,j}, \mathsf{resolve}), X_{r-1,j}, M_{r-2,j}), \mathsf{sign}_T(c, r, j, \mathsf{signed}))$, for some $r > 2$, some j, $X_{r-1,j} = (\mathsf{sign}_1(M_{r-2,j}, r-1, \mathsf{vec_ok}), \ldots, \mathsf{sign}_n(M_{r-2,j}, r-1, \mathsf{vec_ok}))$ and $M_{r-2,j} = (\mathsf{sign}_1(c, r-2, \mathsf{prev_rnd_ok}), \ldots, \mathsf{sign}_n(c, r-2, \mathsf{prev_rnd_ok}))$,

and *stops*. On input wakeup V outputs verify_failed and *stops*.

Lemma 1. *Consider Protocol "sign" of Scheme 1: If T receives $resolve_{r,i}$ and finds $aborted_{s,k} \in abort_set$ for an $s \leq r-2$ then P_k is dishonest.*

Proof. Assume T finds $aborted_{s,k} \in abort_set$ with $s \leq r-2$. Since $s \geq 1$ we have $r \geq 3$, and therefore $resolve_{r,i}$ includes $\mathsf{sign}_k(M_{r-2,k}, r, \mathsf{vec_ok})$, taken from $m_{r-1,k}$. Thus P_k participated in Round $r-1$, and since $r-1 > s$ this means that P_k was still active after having sent $resolve_{s,k}$ to T. Thus P_k is dishonest. □

Theorem 1 (Security of Scheme 1). *Scheme 1 is an asynchronous MPCS with third party T for any $t < n$. It is optimistic on agreement and terminates in $t+2$ rounds if T is not involved, and in $t+4$ rounds in the worst case.*

Proof. (Sketch)

Correct execution, termination of verify and *optimistic on agreement* are all obviously satisfied.

Unforgeability of contract. All variants of a valid contract contain some pieces signed by all signatories, and these signatures exist only if all signatories started the protocol.

Verifiability. The definitions of C_i in the signing protocol for P_i satisfy the conditions checked by V.

No surprises with invalid contracts. Assume an honest P_i started the protocol, decided failed, and some honest verifier V decides signed.

- Assume V decided because of Rule V1. This implies that P_i sent $\mathsf{sign}_i(c, t+2, \mathsf{prev_rnd_ok})$ as part of $m_{t+2,i}$. Thus P_i asked T in Round $t+2$ and received $aborted_{t+2,i}$. According to T's rules this means that there is some signatory $P_{k_{t+1}}$ that received $aborted_{t+1,k_{t+1}}$. Inductively we can show (using T2) that for all rounds $s \in \{1, \ldots, t+1\}$ there is one signatory P_{k_s} that received $aborted_{s,k_s}$. Because of Lemma 1 we know that all signatories P_{k_s} with $s \in \{1, \ldots, t\}$ are dishonest, and as there are at most t dishonest signatories we know that $P_{k_{t+1}}$ must be honest. Since $P_{k_{t+1}}$ asked T in Round $t+1$ it did not send its message for Round $t+2$. Thus no signatory receives all information that is necessary to construct a signed contract. This contradicts our assumption.
- Assume V decides because of Rule V2 or Rule V3, seeing a message $resolve_{r,j}$. Thus P_i asked T in some Round s with $s < r$ and received $aborted_{s,i}$. As P_i is honest we know that $s \geq r-1$ (Lemma 1). But this contradicts Rule T2 of T, i.e., if P_i received $aborted_{s,i}$ then $\mathsf{sign}_T(c, r, i, \mathsf{signed})$ cannot exist. Again we have a contradiction.

Termination of $\mathsf{sign}[]$. For each signatory the protocol proceeds in $t+2$ rounds, and each round terminates, either because the full vector of n messages arrived that allows to enter the next round, or because T is asked and will eventually send an answer (which results in a total number of $t+4$ rounds if T is asked in the last round). □

Remark 2. The protocol is optimistic on agreement only: If P_i starts the protocol but P_k does not then P_i will send $resolve_{1,i}$ to T. In [6] we show that any such protocol can be transformed into a protocol that is optimistic in all cases.

Number of Messages and Rounds. Let C_s and C_b be the costs of a single and a broadcast message, respectively.

In the optimistic case, Protocol "sign" of Scheme 1 runs in $t+2$ rounds where each signatory broadcasts one message to all other signatories, resulting in costs of $(t+2)nC_b$. In the worst case each signatory might have one message exchange with T, resulting in $t+4$ rounds and costs of $(t+2)nC_b + 2nC_s$. If one assumes that each broadcast requires $n-1$ single messages we end up with costs of $((t+2)n(n-1) + 2n)C_s = O(tn^2)C_s$. All broadcast messages of Scheme 1 have length $O(\log n)$, all messages sent to or by T have length $O(n)$; thus we need $O(tn^2 \log n)$ bits only.

Alternatively we can replace each round of n simultaneous broadcasts by 2 rounds and $2(n-1)$ single messages. This will result in $2t+6$ rounds and costs of $((t+2)2(n-1)+2n)C_s = O(tn)C_s$, and still $O(tn^2 \log n)$ bits: Each broadcast round, say, Round r, is implemented by two mini-rounds: in Mini-Round r_1 each signatory P_i with $i > 1$ sends its message to P_1 only. P_1 collects the full vector of n messages, as it would do in the original protocol. If this succeeds it sends this vector to all other signatories in Mini-Round r_2, which concludes Round r. As before, everybody, in particular P_1, can stop waiting any time and can send a message to T. This modification does not affect our proof of security, since

only in the all-honest case messages sent between honest signatories have to be delivered eventually.

According to [13] any asynchronous optimistic multi-party contract signing protocol tolerating $t = n - 1$ requires a number of rounds at least linear in n.

Corollary 1 (Round optimality).
The number of rounds in Protocol "sign" of Scheme 1 is asymptotically optimal for $t = n - 1$.

5 Abuse-Free Asynchronous Multi-party Contract Signing

Scheme 1 is not abuse-free: Consider $n = 2$ and $t = 1$, and assume that P_2 is dishonest. Messages $m_{1,1}$, $m_{1,2}$ and $m_{2,1}$ are delivered. Now the adversary fully controls the result: If he ignores $m_{1,1}$ and $m_{2,1}$ and delivers $resolve_{1,2}$ to honest T then T will decide failed. If he delivers $resolve_{3,2}$ to T then T will decide signed. The message $resolve_{3,2}$ convinces any outside party that T would actually decide signed.

One cannot avoid that the adversary has an advantage in determining whether the protocol ends with signed or with failed, provided all signatories start the protocol. But one *can* avoid that the adversary can prove his ability to force the result signed to an outside party.

In [12,13] this is achieved by introducing and constructing a new type of digital signature scheme, called private contract signatures.

Our solution is conceptually much simpler, as it uses standard cryptographic primitives only: The basic idea is to use Scheme 1 as it is, but let each signatory P_i generate a *new, fresh* pair of secret and public signature key, (sk_i, pk_i), for each execution of Protocol "sign" of Scheme 1. We call the result of such an execution a *pre-contract.* As long as the adversary cannot prove that a certain fresh key belongs to a certain signatory he cannot prove to an outside party the status of the protocol, and hence the protocol is abuse-free.

We call the fresh keys (sk_i, pk_i) *short-term keys,* in contrast to the *long-term keys* used by P_i to generate signatures under its name. Note that by convention (Sect. 2) all the public long-term keys are fixed in advance! Let $\mathsf{SIGN}_i(x)$ denote P_i's signature with its long-term key, and let $\mathsf{sign}_i(x)$ denote P_i's signature with its short-term key sk_i.

Our construction makes use of a chosen-ciphertext secure public-key encryption scheme [9,10]. Only T needs to decrypt messages; let $E_T(r; x)$ denote the encryption of x with T's public key using random string r. We assume that each party knows T's public encryption key.

Scheme 2 (Asynchronous Optimistic Abuse-free MPCS)

Protocol "sign" for honest P_i:

- **Phase 1, Distribution of encrypted certificates:** On input $(\mathsf{sign}, tid, contr)$ signatory P_i generates (sk_i, pk_i) and computes $cert_i :=$

$\mathsf{SIGN}_i(i, pk_i, tid, contr)$ and $ecert_i := E_T(r_i; cert_i)$ for a randomly chosen r_i. P_i distributes (pk_i, $ecert_i$, tid, $contr$) to all signatories.
Each P_i collects the list of all n tuples. It can stop waiting for a tuple any time, in which case it outputs failed and stops.

- **Phase 2, Pre-contract signing:** Protocol "sign" of Scheme 1 is executed to obtain the pre-contract, using the n fresh keys (pk_i, sk_i). The pre-contract to be signed is the concatenation of tid, $contr$, and $((pk_1, ecert_1), \ldots, (pk_n, ecert_n))$.[5]
- **Phase 3, Conversion of pre-contract into real contract:** If P_i gets output signed for the pre-contract then P_i broadcasts $(r_i, cert_i)$ to all other signatories. (This allows any other party to check that $ecert_i$ was an encryption of $cert_i$.)
  If P_i receives input wakeup before having received all n pairs, or if not all pairs match (i.e., there is some j such that $ecert_j \neq E_T(r_j; cert_j)$, or $cert_j$ is not a valid signature by P_j on $(j, pk_j, tid, contr)$) then P_i shows the signed pre-contract to T.
- **Phase 4, Recovery from failed conversion:**
  - If T gets a signed pre-contract it extracts the list $((pk_1, ecert_1), \ldots, (pk_n, ecert_n))$ and tries to decrypt all certificates.
    If this fails, or if some of the decrypted certificates are not valid or not of the proper format, then T signs a "failed" message and sends it back. Otherwise T signs $(tid, contr, cert_1, \ldots, cert_n)$ and sends all certificates and T's signature back. (T's signature certifies that $ecert_j$ was an encryption of $cert_j$ for each j. Note that usually T does not know r_j.)
  - If P_i receives an answer from T it verifies T's signature. If T decided failed then P_i decides failed and stops. Otherwise P_i checks the consistency of the data received from T, and if this succeeds decides signed and stops. If not it decides failed and stops.

Protocol "verify":

If P_i wants to show a signed contract to verifier V it sends to V the pre-contract from Phase 2, signed relative to short-term keys $pk_1, \ldots, pk_n$ and (a) all n pairs $(r_i, cert_i)$ or (b) the vector $(tid, contr, cert_1, \ldots, cert_n)$, signed by T. V outputs signed if all information matches, and *stops*.

On input wakeup V outputs verify_failed and *stops*.

Theorem 2 (Security of Scheme 2).

Scheme 2 is an asynchronous MPCS with third party T for any $t < n$. It is optimistic in all cases and terminates in $t + 4$ rounds if T is not involved, and in $t + 6$ rounds in the worst case.

Proof. (Sketch) We prove only three properties:

[5] Note that this pre-contract is *not* the real contract yet, and does not prove anything to an outside party: we will show that the adversary can generate valid looking pre-contracts himself, without any interaction with the honest signatories.

"Correct execution:" In the all-honest case the channels between signatories ensure eventual delivery and *authenticity* (Sect. 2). Thus the adversary cannot disturb the protocol, and it terminates with decision signed.

"Unforgeability:" Assume some honest P_i did not want to sign, but a verifer V outputs signed. Th. 1 implies that pre-contracts are unforgeable relative to the short-term keys, and Protocol "verify" requires that one of the short-term keys was certified by P_i, which contradicts our assumption. (Note that the set of signatories is fixed in advance; see Sect. 2.)

"No surprises with invalid contracts:" Assume an honest P_i started the protocol, decided failed, and some honest verifier V decides signed.

Th. 1 implies that there was a run of Protocol "sign" of Scheme 1 that yielded a signed pre-contract. P_i participated in this run as one of the short-term keys used was certified by P_i relative to the original $(tid, contr)$, and P_i certifies only fresh keys. Therefore P_i obtained a signed pre-contract as well. As P_i finally decided failed it must have obtained "failed" from T in Phase 4, which means that T was not able to successfully decrypt and check all certificates.

Since T does not distribute inconsistent decisions V accepted the contract because of the result of a successful Phase 3. Thus there must be n pairs $(r_j, cert_j)$ that are valid certificates and consistent with the signed contract. But T was not able to obtain the corresponding n certificates $cert_i$. Since T is honest this contradicts the correctness requirement of the encryption scheme. □

Remark 3. Phase 1 does not guarantee that $ecert_i$ actually contains a valid certificate $cert_i$, i.e., it might happen that there are invalid certificates but Phase 2 is started and terminates successfully.

At first glance this might look dangerous, but actually it is not: If at least one $ecert_i$ is invalid—i.e., decryption does not yield a certificate $cert_i$ on pk_i signed by P_i— then no party will obtain the real contract.

Theorem 3 (Abuse freeness). *Scheme 2 is abuse-free.*

Proof. (Sketch) We prove this property by contradiction: Assume that at a certain point in time the adversary has full control over the contract and can convince an outside party that he can enforce result signed.

Since the adversary has full control still both results are possible. Therefore no party can show a valid pre-contract, as the information contained in it would determine the result of the contract signing. This follows from Th. 2: If one signatory had a signed pre-contract then any other signatory would obtain it as well, and T could unambiguously decide by evaluating this pre-contract. Therefore no honest signatory will enter Phase 3, i.e., no honest signatory will open any encryption.

In this situation we can simulate the adversary's view of the protocol: The simulator generates fresh keys (sk_i, pk_i), exactly like the honest signatories did. But instead of encrypting certificates $cert_i$ on these keys, the simulator encrypts random data of the same length. Since the encryption scheme is secure against adaptive chosen ciphertext attacks the resulting ciphertexts $ecert_i$ are indistinguishable from encrypted certificates. This simulation works as long as none of

the ciphertexts must be opened—which is the case as no valid pre-contract exists, yet. Thus, any interaction between the adversary and an outside party can be simulated without any interaction with the honest signatories. Hence, the outside party gets no information about whether the honest signatories actually want to sign a contract, which contradicts our assumption. □

Acknowledgments: We thank *Juan Garay, Birgit Pfitzmann, Matthias Schunter* and *Michael Steiner* for interesting discussions.

References

1. N. Asokan, B. Baum-Waidner, M. Schunter, M. Waidner: Optimistic Synchronous Multi-Party Contract Signing; IBM Research Report RZ 3089 (#93135), Zürich, December 1998.
2. N. Asokan, M. Schunter, M. Waidner: Optimistic Protocols for Multi-Party Fair Exchange; IBM Research Report RZ 2892, Zürich, November 1996.
3. N. Asokan, M. Schunter, M. Waidner: Optimistic Protocols for Fair Exchange; 4th ACM Conf. on Computer and Communications Security, Zürich, April 1997, 6–17.
4. N. Asokan. V. Shoup, M. Waidner: Optimistic Fair Exchange of Digital Signatures; Eurocrypt '98, LNCS 1403, Springer-Verlag, Berlin 1998, 591–606.
5. B. Baum-Waidner, M. Waidner: Asynchronous Optimistic Multi-Party Contract Signing; IBM Research Report RZ3078 (#93124), Zürich, November 1998.
6. B. Baum-Waidner, M. Waidner: Round-optimal and Abuse-free Optimistic Multi-Party Contract Signing; IBM Research Report RZ3199 (#93245), Zürich, Jan 00.
7. M. Ben-Or, O. Goldreich, S. Micali, R. L. Rivest: A Fair Protocol for Signing Contracts; IEEE Transactions on Information Theory 36/1 (1990) 40–46.
8. M. Blum: Three Applications of the Oblivious Transfer; Department of Electrical Engineering and Computer Sciences, University of California at Berkley, September 18, 1981.
9. R. Cramer, V. Shoup: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack; Crypto '98, LNCS 1462, Springer-Verlag, Berlin 1998, 13–25.
10. D. Dolev, C. Dwork, M. Naor: Non-Malleable Cryptography; 23rd Symposium on Theory of Computing (STOC) 1991, ACM, New York 1991, 542–552.
11. S. Even, Y. Yacobi: Relations Among Public Key Signature Systems; Technical Report Nr. 175, Computer Science Department, Technion, Haifa, Israel, 1980.
12. J. Garay, M. Jakobsson, P. MacKenzie: Abuse-free Optimistic Contract Signing; Crypto '99, LNCS 1666, Springer-Verlag, Berlin 1999, 449–466.
13. J. Garay, P. MacKenzie: Abuse-free Multi-party Contract Signing; Intern. Symp. on Distr. Comput. (DISC '99), LNCS 1693, Springer-Verlag, Berlin 1999, 151–165.
14. S. Goldwasser, S. Micali, R. Rivest: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks; SIAM J. Comput. 17/2 (1988) 281–308.
15. A. Menezes, P. van Oorschot, S. Vanstone: Handbook of Applied Cryptography; CRC Press, Boca Raton 1997.
16. S. Micali: Certified E-Mail with Invisible Post Offices; 1997 RSA Conference.
17. B. Pfitzmann, M. Schunter, M. Waidner: Optimal Efficiency of Optimistic Contract Signing; ACM PODC '98, Puerto Vallarta 1998, 113–122.
18. M. Rabin: Transaction Protection by Beacons; Journal of Computer and System Sciences 27/ (1983) 256–267.

On the Centralizer of a Finite Set *

Juhani Karhumäki and Ion Petre

Department of Mathematics, University of Turku and
Turku Centre for Computer Science (TUCS)
Turku 20014, Finland
{karhumak,ipetre}@cs.utu.fi

Abstract. We prove two results on commutation of languages. First, we show that the maximal language commuting with a three element language, i.e. its *centralizer*, is rational, thus giving an affirmative answer to a special case of a problem proposed by Conway in 1971. Second, we characterize all languages commuting with a three element code. The characterization is similar to the one proved by Bergman for polynomials over noncommuting variables, cf. Bergman, 1969 and Lothaire, 2000: A language commutes with a three element code X if and only if it is a union of powers of X.

1 Introduction

Very little, or in fact almost nothing, seems to be known on solutions of language equations, the exception being very special equations with two operations characterizing rational languages, cf. [7] and [10] for some extensions. Even the most basic equation, namely the commutation $XY = YX$, is poorly understood. On the other hand, it proposes several natural and apparently very difficult combinatorial problems.

It was almost 30 years ago when Conway proposed such a problem, asking whether the maximal set commuting with a given rational set is rational, see [6]. The problem remained unanswered up-to-date, even for finite sets. Even worse, it seems to be unknown whether the centralizer of a finite set is recursive, or even recursively enumerable. A related problem asking whether any decomposable rational language L, i.e. a rational language having the decomposition $L = XY$ for some languages $X, Y \neq \{1\}$, is decomposable via rational languages, is much simpler, as shown in [6], cf. also [8], [13], and [3].

Another related problem is to search for a characterization of all languages commuting with a given rational or finite set. In the case of multisets, i.e. polynomials over noncommuting variables and with rational coefficients, this problem has an elegant solution due to Bergman [1]: Two polynomials $p(\overline{x})$ and $q(\overline{x})$ commute if and only if they are linear combinations of powers of a common polynomial $t(\overline{x})$. A similar result holds also for noncommutative formal power series, see [5].

* The authors acknowledge the support from the Academy of Finland under project 44087.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 536–546, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Recently, both of the above problems have been solved for two element sets in [4]. In this case, Conway's problem has an affirmative answer and moreover, the binary sets possess a Bergman type of characterization: Any set commuting with a two element set X is a union of powers of X (or just a union of powers of a primitive word t, if $X \subseteq t^*$, for some word t). On the other hand, as was pointed out also in [4], no similar characterization can be achieved for four element sets, in general.

These problems were also considered in the case of codes, in [14]. They have been completely and affirmatively answered if X is a prefix code, i.e. no word is a prefix of another. Moreover, it was proved that for a prefix code X, its centralizer is always X^*, and the same was conjectured to be true for all codes. This, however, remains an unsolved - and difficult - problem.

In this paper, we continue the study of these two problems, considering a special case of three element sets. We answer to Conway's problem affirmatively in this case, and show that Bergman type of characterization, conjectured in [14], holds for three element codes. Our new idea of considering these problems as equations on languages, combined with the techniques of [4], [13], and [14], gives a new insight on the problem. What is encouraging is that we do not see any reason why this approach cannot be extended to more general cases. We also point out that in general, the centralizer of any finite set is in *Co-RE*.

The paper is organized as follows. In Section 2, we fix the terminology and discuss the background of these problems. Several basic results needed in later considerations, as well as two general results on the centralizer of a finite language, are proved in Section 3. Section 4 is devoted to a solution of Conway's problem for three element sets and in Section 5 we prove the characterization for languages commuting with a three element code. Several open problems are proposed in Section 6. Due to space constrains, the proofs of some results are only sketched here. They can be found in full length in [9].

2 Preliminaries and Background

In this section we fix our terminology, and recall several known results related to this work. For further details in Combinatorics on Words, we refer to [2].

For two words $u, v \in \Sigma^*$, we say that u is a *prefix* of v if $v = uw$, for some $w \in \Sigma^*$, and write $u \leq v$, and $u = vw^{-1}$; u is a *proper prefix* of v if both u and w are nonempty words. We say that u is a *suffix* of v if $v = wu$, for some $w \in \Sigma^*$, in which case we write $u = w^{-1}v$. These notations extend in a natural way to languages: for two languages $L_1, L_2 \subseteq \Sigma^*$, we denote

$$L_1^{-1}L_2 = \{u_1^{-1}u_2 \mid u_1 \in L_1, u_2 \in L_2\},$$
$$L_1L_2^{-1} = \{u_1u_2^{-1} \mid u_1 \in L_1, u_2 \in L_2\}.$$

For a finite language F we define the two parameters

$$l_F = \min_{u \in F} |u|, \quad L_F = \max_{u \in F} |u|,$$

where $|u|$ denotes the length of the word u. We say that F is *periodic* if there is a word u such that $F \subseteq u^*$.

It is an elementary property on commutation of languages that for any subset $L \subseteq \Sigma^*$, there is a unique maximal language commuting with it and, moreover, it can be easily proved that it is a monoid. We will call it the *centralizer* of L and denote it as $\mathcal{C}(L)$. Equivalently, one can define the centralizer of L as the union of all sets commuting with L.

We say that a language $L \subseteq \Sigma^*$ is a *code* if the monoid L^* is free. Equivalently, L is a code iff any equality

$$x_1 x_2 \dots x_m = y_1 y_2 \dots y_n, \qquad m, n \geq 0, x_i, y_j \in L$$

implies $n = m$ and $x_i = y_i$, for all $1 \leq i \leq m$.

Let Σ be a finite alphabet, and Ξ a set of unknowns in one-to-one correspondence with a set of nonempty words $X \subseteq \Sigma^*$, say $\xi_i \leftrightarrow x_i$, for some fixed enumeration of X. A (constant-free) *equation* over Σ with Ξ as the set of unknowns is a pair $(u, v) \in \Xi^* \times \Xi^*$, usually written as $u = v$. The subset X *satisfies* the equation $u = v$ if the morphism $h : \Xi^* \to \Sigma^*$, $h(\xi_i) = x_i$, for all $i \geq 0$, is such that $h(u) = h(v)$. These notions extend in a natural way to *systems of equations.*

We define the *dependence graph* of a system of equations S, as the nondirected graph G, whose vertices are the elements of Ξ, and whose edges are the pairs $(\xi_i, \xi_j) \in \Xi \times \Xi$, with ξ_i and ξ_j appearing as the first letters of the left and right handsides of some equation of S, respectively.

The following basic result on combinatorics of words, cf. [2], is very useful and efficient in our later considerations.

Lemma 1 *(Graph Lemma) Let S be a system and let $X \subset \Sigma^+$ be a subset satisfying it. If the dependence graph of S has p connected components, then there exists a subset F of cardinality p such that $X \subseteq F^*$.*

Note that in Graph Lemma it is crucial that all words are nonempty.

Concerning the commutation of languages, we will be interested in the following two problems, cf. [6] and [13], respectively:

Conway's Problem: Is the centralizer of a rational language, rational?

BTC-Problem: For a given finite set $X \subseteq \Sigma^*$, is it true that for any set Y commuting with X, there exists a set $V \subseteq \Sigma^+$ and sets I, J of nonnegative integer indices, such that

$$X = \bigcup_{i \in I} V^i \quad \text{and} \quad Y = \bigcup_{j \in J} V^j \ ? \tag{1}$$

Note that if X and Y satisfy (1), then they commute. The statement of the BTC-problem is the same as the one proved by Bergman in [1] to characterize the commutation of two polynomials over noncommuting variables. The abbreviation BTC comes from there: Bergman Type Characterization.

First results on Conway's problem were achieved in [14], where it has been proved that the answer is affirmative for all prefix codes. Recently, in [4], the same was achieved for all binary sets. The BTC-Problem was also solved affirmative in these two cases in the same two papers, respectively. Moreover, it was shown in [4] that the BTC-Problem does not have a positive answer in general, or even in the case of four element arbitrary sets. A simple counterexample is the set $X = \{a, ab, ba, bb\}$, which commutes with $X \cup X^2 \cup \{bab, bbb\}$.

3 Auxiliary Results

In this section we prove some lemmata needed in our later considerations, as well as give some general properties of the centralizer.

In both of the following lemmata, we will consider the maximal languages with respect to component-wise inclusion, satisfying some given relations. In each case, their uniqueness is clear.

Lemma 2 *Let F_1, F_2, F_3, and G be finite languages. If the relation*

$$F_1 + F_2XG = F_3 + YG, \tag{2}$$

is satisfiable, then the maximal solution (X, Y) is such that $Y = F_2X + H$, for some finite language H.

Proof. Clearly, $F_2X \subseteq Y$, since Y is maximal and we could add to Y the elements of F_2X preserving the relation (2).

On the other hand, consider $w \in G$, and $y \in Y$, with $|y| > \max(L_{F_1}, L_{F_2})$. Then, $yw \in YG$, and hence, $yw \in F_2XG$:

$$yw = uxv, \quad u \in F_2, x \in X, v \in G,$$

which implies that $y = uy_1$, for some $y_1 \in \Sigma^*$. We can add y_1 to X (and hence, y to F_2X) and F_2y_1 to Y, preserving the equality. Hence $Y = F_2X + H$, where $H = F_1G^{-1}$.

It is also useful to note that if the equation (2) has solutions, then $F_3 \subseteq F_1 + YG$, and $F_1 = F_1'G + F_1''$, with $F_1'' \subseteq F_3$. In other words, the maximal languages X and Y satisfying (2) actually satisfy $F_1'G + F_2XG = YG$, too. ■

Lemma 3 *Let F_1, F_2, G, and H be finite languages. If the language equations*

$$F_1X = XF_2, \quad G + F_1Y = H + YF_2,$$

are satisfiable, then their maximal solutions X and Y are such that $Y = X + A$ for some finite language A.

Proof. Consider the equations $u^{-1}(G + F_1Y) = u^{-1}(H + YF_2)$, for all $u \in \Sigma^*$, with length $|u| = 1 + \max(L_G + L_H)$. Since the word u is longer than the words of G and H, we obtain the system $u^{-1}(F_1Y) = u^{-1}(YF_2)$. But then, the same

system is satisfied also by X. Since the maximal solution of a given linear system of equations can be readily seen to be unique, we obtain that $u^{-1}X = u^{-1}Y$, for all u such that $|u| = 1 + \max(L_G, L_H)$. Furthermore, clearly $X \subseteq Y$, since we can add X to Y, and the relation satisfied by Y is still valid. Hence, $Y = X + A$, for some finite language A. It is useful to observe that $A = F_1^{-1}H + GF_2^{-1}$. ■

When computing the centralizer of a language, we will always assume that the language does not contain the empty word since otherwise *Conway's problem* is trivial. Indeed, the centralizer of such a language is always Σ^*, as noticed also in [4]. Also remark that in this case the *BTC-problem* has a negative answer; to see this, it is enough to consider any language L such that $L^* \neq \Sigma^*$, and notice that $L' = L \cup \{1\}$ commutes with Σ^*, but they are not unions of powers of a third language.

In general, very little is known about the centralizer of a finite language. Even much weaker questions than Conway's question seem to be unanswered, namely it is not known whether a centralizer of a finite language is recursive or even recursively enumerable. We are able to show only that its complement is always recursively enumerable.

Theorem 4 *For any finite set, the complement of its centralizer is a recursively enumerable language.*

4 Conway's Conjecture for Three Element Sets

We consider now Conway's problem for finite languages. To start with, let

$$FX = XF, \tag{3}$$

be the language equation for a given finite language F. Obviously, the language F can be uniquely written as

$$F = u_1F_1 + u_2F_2 + \cdots + u_nF_n, \tag{4}$$

such that the following conditions are satisfied:

(i) $1 \in F_i$, for all $i = 1, \ldots, n$.
(ii) $u_i \notin \mathrm{Pref}(u_j)$, for any $i \neq j$.

We will say that (4) is the *prefix decomposition* of F.

Example 1. The prefix decomposition of $F = \{a, aa, b, bab\}$ is $F = a \cdot \{1, a\} + b \cdot \{1, ab\}$.

We say that two nonempty words are *incomparable* if neither of them is a proper prefix of the other.

Using this notion to refine the equation $FX = XF$, we can prove that the centralizer of a finite language is rational, provided that one additional condition is satisfied: there is a word in F such that it is incomparable with the other words of F (for example, all prefix languages satisfy this condition). Using this result, we then prove Conway's conjecture for three element sets.

Theorem 5 *If F is a finite language containing a word which is incomparable with the other words of F, then the centralizer of F is rational and effectively computable.*

Proof. Consider the prefix decomposition of F: $F = u_1F_1 + \cdots + u_nF_n$, and let $\mathcal{C}(F)$ be its centralizer. Then clearly, $\mathcal{C}(F) = u_1X_1 + \cdots + u_nX_n + X_0$, with

$$X_0 = \{\, x \in \mathcal{C}(F) \mid |x| < L_F \},$$

since the equation (3) implies that all long enough words of $\mathcal{C}(F)$ have as a prefix an element of F. Then, the equation (3) implies that

$$F_i\mathcal{C}(F) = X_iF + u_i^{-1}(X_0F), \quad \text{for all } 1 \le i \le n. \tag{5}$$

Now, the hypothesis implies that one of the sets F_i, say F_1, contains only the empty word. We then obtain from the system (5) that

$$F_iX_1F + F_i(u_1^{-1}(X_0F)) = X_iF + u_i^{-1}(X_0F), \quad \text{for all } 2 \le i \le n.$$

By Lemma 2, this means that $F_iX_1 + H_i = X_i$, for all $2 \le i \le n$, and for some finite sets $H_1, \ldots, H_n$. So,

$$\begin{aligned}\mathcal{C}(F) &= X_0 + u_1X_1 + u_2(H_2 + F_2X_1) + \cdots + u_n(H_n + F_nX_1)\\ &= X_0 + u_2H_2 + \cdots + u_nH_n + (u_1 + u_2F_2 + \cdots + u_nF_n)X_1\\ &= H + FX_1,\end{aligned}$$

and this, together with the first equation of the system (5), further implies that

$$u_1^{-1}(X_0F) + X_1F = H + FX_1,$$

where $H = X_0 + \sum_{i=2}^{n} u_iH_i$. Obviously, X_1 is maximal since $\mathcal{C}(F)$ is so. By Lemma 3, we then obtain that $X_1 = A + \mathcal{C}(F)$, for some finite set A, and finally,

$$\mathcal{C}(F) = H + F(\mathcal{C}(F) + A) = H + FA + F\mathcal{C}(F),$$

i.e., $\mathcal{C}(F)$ is rational. Moreover, $\mathcal{C}(F)$ is of the form $\mathcal{C}(F) = F^*G$, with G a finite language. The above equations can be further refined to prove that $L_G \le 3L_F - 2l_F$. Consequently, for a given F, one can effectively find G and hence, also $\mathcal{C}(F)$. The theorem is thus proved. ■

It might be possible to refine this method for the general case. For this, we need to solve the system (5) in the case when all F_i's are different from $\{1\}$. One way to do this is to consider the prefix decomposition of the finite sets appearing as coefficients in the left hand side of (5) and refine the right hand side accordingly. It seems that in this way, we either cycle (case which we can solve as in the proof of the next theorem), or we eventually obtain a case similar to the one in the previous theorem. However, what we can do now is to settle completely the three element case, as shown in the next theorem.

Theorem 6 *The centralizer of a three element set is rational and effectively computable.*

Proof. By Theorem 5, the only case to be considered is that of a finite set F of the form $F = \{u, uv, uw\}$. In this case, $\mathcal{C}(F) = uX_1 + X_0$, with X_0 containing only prefixes of u. The equation (3) can now be rewritten as $(1 + v + w)\mathcal{C}(F) = X_1F + u^{-1}(X_0F)$, and further in the form

$$(u + vu + wu)X_1 + (u^{-1}F)X_0 = X_1F + u^{-1}(X_0F).$$

We repeat the same reasoning with X_1 by taking the prefix decomposition of the set $\{u, vu, wu\}$. If the words of F are not powers of a same word (which is solved in [4] and [13]), then after a finite number of steps we obtain an equation of the form

$$GX_n + H_1 = X_nF + H_2,$$

where all G, H_1, H_2 are finite languages, and G is a three word language such that at least two of its words start differently. Moreover, we have that $\mathcal{C}(F) = u^nX_n + X_0'$, with X_0' finite.

We consider now the equation $GY = YF$, and we can prove that its maximal solution is rational, using the same techniques as in the proof of Theorem 5. We then use Lemma 3 to obtain that $X_n = Y + A$, for some finite language A. Hence, X_n is also rational. Then the centralizer of F is itself rational, which concludes the proof. ■

We can also specify the form of the centralizer of a three element set. Similarly as in the proof of the Theorem 5, one can prove that $X_n = G^*A + B$, with A and B finite languages. Then, $X = u^nG^*A + B'$, and knowing that G has been obtained from F by shifting n u-s from its left to the right, we obtain that $X = F^*A' + B'$, with A' and B' finite languages.

5 The Solution of the BTC-Problem for Three Element Codes

In this section we characterize all sets commuting with a given three element code. The characterization resembles that of Bergman for polynomials over non-commuting variables. Namely, we prove that any set of words commuting with a three element code X is a union of powers of X. The same condition holds for singletons, and also for two word languages, as proved in [4]. On the other hand, this is not valid anymore for four element sets, as we already mentioned.

We first recall a result proved in [4]:

Lemma 7 *Let L be a language and $\mathcal{C}(L)$ its centralizer. For any $x \in L$ and $z \in \mathcal{C}(L)$, there exists an infinite word $u \in \mathcal{C}(L)^\omega$ such that $zx^\omega = u$.*

We will also need the following lemma, proved in [4] and [14]:

Lemma 8 *Let $X \subseteq \Sigma^*$ be a code such that its centralizer is X^*, and let $Y \subseteq \Sigma^*$ be a language commuting with X. If $Y \cap X^n \neq \emptyset$, for some $n \geq 0$, then $X^n \subseteq Y$.*

The first step towards our goal is to prove that the centralizer of a three word code F is F^*. The next lemma will be the main tool to achieve this, and the most difficult part of this paper.

Lemma 9 *Let F be a nonperiodic three word set such that none of its elements is a prefix of the other two, and let $\mathcal{C}(F)$ be its centralizer. Then, all words of $\mathcal{C}(F) - \{1\}$ have as a prefix a word from F.*

Proof. Let F be the language $\{u, v, w\}$ satisfying the conditions of the lemma. As we already observed, all long enough words of $\mathcal{C}(F)$ have as a prefix a word from F. Let us consider the set of those "short words" of $\mathcal{C}(F)$ which do not have as a prefix a word from F. The claim of the lemma is that this set, say X_0, contains only the empty word. Obviously, $1 \in X_0$, so let us assume that there is a nonempty word $x \in X_0$. By Lemma 7, we have that

$$\begin{aligned} xu^\omega &= \alpha_1\alpha_2 \ldots \alpha_n \ldots \\ xv^\omega &= \beta_1\beta_2 \ldots \beta_n \ldots \\ xw^\omega &= \gamma_1\gamma_2 \ldots \gamma_n \ldots \end{aligned} \tag{6}$$

for some $\alpha_i, \beta_i, \gamma_i \in F$, $i \geq 1$. Let us denote $A = \{\alpha_1, \beta_1, \gamma_1\}$.

If the cardinal of A is 3, that is to say, $A = F$, then by Graph Lemma on the system (6), we conclude that F is periodic: a contradiction. If A is a singleton, e.g. $A = \{u\}$, then, as x is from X_0, we have that x is a prefix of u: $u = xt$, with $t \neq 1$. Hence, we conclude again by Graph Lemma on the set of unknowns $\{t, u, v, w\}$ of (6): F must be periodic.

Assume now that A has cardinality 2. In this case, one can prove that X_0 is totally ordered by the prefix relation. If in the system (6), $\alpha_1 = u$ or $\beta_1 = u$, then $u = xt$, for some $t \neq 1$, and we can conclude again by Graph Lemma. In the remaining the case when $\alpha_1 = \beta_1 = v$, i.e.

$$xu = vy_1, \quad xv = vy_2, \quad xw = uy_3,$$

for some $y_1, y_2, y_3 \in \mathcal{C}(F)$, the proof strategy is to distinguish two cases: $y_2 \in X_0$ (obtaining $y_2 = x$), or $y_2 \notin X_0$ (obtaining $w \leq y_2$), and aiming to apply the Graph Lemma to conclude that F must be periodic. Due to space limitations, we skip this part of the proof, referring to [9] for the complete proof. ■

We can prove now that F^* is the maximal set commuting with F, for any three word code.

Theorem 10 *The centralizer of a three word code F is F^*.*

Proof. Let $\mathcal{C}(F)$ be the centralizer of F. We distinguish three cases, depending on the prefix decomposition of F, as defined in Section 2.

I. $F = u_1 + u_2 + u_3$, and none of the words of F is a prefix of another one. Then, using Lemma 9, we have that $\mathcal{C}(F) = u_1X_1 + u_2X_2 + u_3X_3 + 1$. From the equation $F\mathcal{C}(F) = \mathcal{C}(F)F$, we obtain the following equations:

$$\mathcal{C}(F) = X_1F + 1, \quad \mathcal{C}(F) = X_2F + 1, \quad \mathcal{C}(F) = X_3F + 1,$$

implying that $X_1 = X_2 = X_3$, and hence, $\mathcal{C}(F) = FX_1 + 1$. But then, the system is equivalent to $FX_1 + 1 = X_1F + 1$, which, in turn, is equivalent to $FX_1 = X_1F$. Thus, $X_1 = \mathcal{C}(F)$ and furthermore, $\mathcal{C}(F) = F\mathcal{C}(F) + 1$, which, as is well-known, see [7], implies the claim.

II. $F = u(1 + v) + w$, and u and w are not prefixes of one another. Similarly as above, we obtain that $\mathcal{C}(F) = uX_1 + wX_2 + 1$, and then, the equations

$$(1 + v)\mathcal{C}(F) = X_1F + 1 + v, \quad \mathcal{C}(F) = X_2F + 1.$$

This implies that $(1+v)X_2F+1+v = X_1F+1+v$, and so, $(1+v)X_2 = X_1$. Indeed, we can add X_1F to the left hand side, or $(1 + v)X_2F$ to the right hand side, preserving the equality, and then we use the maximality. Hence, $\mathcal{C}(F) = FX_2+1$, and the above system is equivalent now to $FX_2 + 1 = X_2F + 1$. We conclude as in the previous case.

III. $F = u_1 + u_2 + u_3$, with u_1 a prefix of both u_2 and u_3. Equivalently, F is of the form $F = \{u, uv, uw\}$. Assume that F^* is a proper subset of $\mathcal{C}(F)$, and let x be minimal with respect to the length in $\mathcal{C}(F) - F^*$.

Claim 1: $xu = ux$.

Proof of claim 1. Assume the contrary and set $x_1 = x$. Then we define the sequence $(x_i)_{i\leq n}$, for some $n \geq 1$, such that $x_iu = ux_{i+1}$, for all $1 \leq i \leq n - 1$, and either $x_nu = ux_i$, with $1 \leq i \leq n$, or $x_nu = ty$, with $t \in \{uv, uw\}$, and $y \in \mathcal{C}(F)$. The first case is impossible since we obtain that $x_1 = \cdots = x_n$, and hence, $xu = ux$. In the second case we obtain that $|y| < |x|$ and thus, $y \in F^*$. The conclusion is that $xu^n = u^{n-1}ty \in F^*$. Similarly we can prove that there is $m \geq 1$ such that $u^mx \in F^*$. But F is a code, i.e. F^* is free, and then, Schützenberger's criterium of a free monoid implies that $x \in F^*$, cf. [11]. This is impossible and the claim is thus proved.

As a consequence of the claim we obtain that x is the only minimal element in $\mathcal{C}(F) - F^*$. Let us now assume that $|v| \leq |w|$.

It is important to observe that neither v, nor w can commute with u since F is a code.

Claim 2: There is $\alpha \in F^*$ such that $x\alpha \in F^*$.

Proof of claim 2. Assume again the contrary and consider the word xuv. We prove first that there exists $n \geq 0$ such that

$$xvu^n = u^nwx. \tag{7}$$

If $xuv = uvy$, for some $y \in \mathcal{C}(F)$, we have that $|x| = |y|$, and so, either $x = y$, which is impossible since it implies that $uv = vu$, or $y \in F^*$, and so, $xuv \in F^*$, which is again a contradiction.

If $xuv = uwy$, we obtain that $|y| \leq |x|$, and due to our assumption, we must have $y = x$, and $xv = wy$, which is what we wanted to prove first.

Finally, if $xuv = uy$, then $y = xv$. In this case, we consider the word $y_1 = y$, and then, the word $y_1 u = xvu \in \mathcal{C}(F)F$. We have $y_i u = uy_{i+1}$, for all $1 \leq i \leq n-1$, for some $n \geq 1$, and either $y_n u = uy_i$, for some $1 \leq i \leq n$, or $y_n u = uvt$, or $y_n u = uwt$, for some $t \in \mathcal{C}(F)$. In the first two cases we obtain that $uv = vu$, which is impossible. In the third one we obtain $xvu^n = u^n wx$, for some $n \geq 1$.

Let us also consider the word xuw. Similarly as above (or using the symmetry as an argument), one can prove that there is $m \geq 0$ such that

$$xwu^m = u^m vx. \tag{8}$$

Without loss of generality, we can assume that $m \leq n$. If $x \leq u^m$, then $u^m = xy$, with $y \neq 1$, $yu = uy$, and from (7) and (8) we derive that $u^{n-m}yw = vu^{n-m}y$, $yv = wy$, and $yu = uy$. Using Graph Lemma on these three relations, on the set of unknowns $\{y, u, v, w\}$, we get now a contradiction.

If $u^n \leq x$, then $x = u^n y$, with $y \neq 1$, $uy = yu$, and we obtain $yv = wy$, $u^{n-m}yw = vu^{n-m}y$, $yu = uy$, and again this gives the commutativity of F.

Finally, if $u^m \leq x \leq u^n$, we have $u^n = xy$, and $x = u^m z$, with $y, z \neq 1$, $yu = uy$, and $zu = uz$. We derive from (7) and (8) that $xvu^n = u^n wx = (yx)w(u^m z) = y(xwu^m)z = yu^m vxz$, implying that $uv = vu$, which is impossible. This completes the proof of claim 2.

Claim 3: There is $\beta \in F^*$ such that $\beta x \in F^*$.

Proof of claim 3. This is similar to the proof of claim 2 above.

Since F is a code, from the claims 2 and 3, using Schützenberger's criterium of a free monoid, we obtain that $x \in F^*$. This is impossible: x was chosen in $\mathcal{C}(F) - F^*$.

The conclusion is that $\mathcal{C}(F) \subseteq F^*$, and thus, $\mathcal{C}(F) = F^*$. ■

Now, we are ready for the second main result of this paper. We characterize all the sets commuting with a three element code.

Theorem 11 *If F is a three word code, then any set commuting with F is a union of powers of F.*

Proof. This is immediate now, using Theorem 10 and Lemma 8. ■

6 Final Remarks

We have continued the research on the commutation relation $XY = YX$ for languages, initiated in [14], [13], and [4]. Our results settle some basic problems for three element sets, and at the same time give indications that these problems are very difficult, in general. Indeed, there remain many challenging open problems:

Problem 1 *Does the Bergman type of characterization hold for all three element sets?*

Problem 2 *Does the Bergman type of characterization hold for all codes?*

Problem 3 *Does the Conway's problem have an affirmative answer for all finite codes?*

Problem 4 *Is the centralizer of a code C always C^*?*

Problem 5 *For a finite set F we define its root as the set $\mathcal{R}(F)$ with the minimal number of elements such that F is a power of $\mathcal{R}(F)$. Does there exist a finite set $F \subseteq \Sigma^*$ such that its centralizer is different from $\mathcal{R}(F)^*$ and Σ^*?*

Problem 6 *Is the centralizer of a finite (or rational) set always: a) recursively enumerable, b)recursive, c)rational?*

By Lemma 8, an affirmative answer to Problem 4 would imply that for Problem 2. Also note that if the centralizer is defined as the maximal semigroup commuting with a given X, then the Problem 5 has an affirmative answer, cf. [4].

References

1. G. Bergman, Centralizers in free associative algebras, *Transactions of the American Mathematical Society* 137: 327–344, 1969.
2. C. Choffrut, J. Karhumäki, Combinatorics on Words. In G. Rozenberg, A.Salomaa, eds., *Handbook of Formal Languages*, vol 1: 329-438, Springer-Verlag, 1997.
3. C. Choffrut, J. Karhumäki, On Fatou properties of rational languages, TUCS Technical Report 322, http://www.tucs.fi/, 1999.
4. C. Choffrut, J. Karhumäki, N. Ollinger, The commutation of finite sets: a challenging problem, TUCS Technical Report 303, http://www.tucs.fi/, 1999, submitted to a special issue of Theoret. Comput. Sci. on Words.
5. P. M. Cohn, Centralisateurs dans les corps libres, in J. Berstel (Ed.), *Séries formelles*: 45–54, Paris, 1978.
6. J. H. Conway, *Regular Algebra and Finite Machines*, Chapman Hall, 1971.
7. S. Eilenberg, *Automata, Languages and Machines*, Academic Press, 1974.
8. L. Kari, On insertion and deletion in formal languages, Ph.D. Thesis, University of Turku, 1991.
9. J. Karhumäki, I. Petre, *On the centralizer of a finite set*, Technical Report 342, TUCS, http://www.tucs.fi/, 2000.
10. E. Leiss, *Language Equations*, Springer, 1998.
11. M. Lothaire, *Combinatorics on Words* (Addison-Wesley, Reading, MA., 1983).
12. M. Lothaire, *Combinatorics on Words II*, to appear.
13. A. Mateescu, A. Salomaa, S. Yu, On the decomposition of finite languages, Technical Report 222, TUCS, http://www.tucs.fi/, 1998.
14. B. Ratoandromanana, Codes et motifs, *RAIRO Inform, Theor.*, 23(4): 425-444, 1989.

On the Power of Tree-Walking Automata

Frank Neven[1,⋆] and Thomas Schwentick[2]

[1] Limburgs Universitair Centrum
[2] Johannes Gutenberg-Universität Mainz, Institut für Informatik

Abstract. Tree-walking automata (TWAs) recently received new attention in the fields of formal languages and databases. Towards a better understanding of their expressiveness, we characterize them in terms of transitive closure logic formulas in normal form. It is conjectured by Engelfriet and Hoogeboom that TWAs cannot define all regular tree languages, or equivalently, all of monadic second-order logic. We prove this conjecture for a restricted, but powerful, class of TWAs. In particular, we show that 1-bounded TWAs, that is TWAs that are only allowed to traverse every edge of the input tree at most once in every direction, cannot define all regular languages. We then extend this result to a class of TWAs that can simulate first-order logic (FO) and is capable of expressing properties not definable in FO extended with regular path expressions; the latter logic being a valid abstraction of current query languages for XML and semi-structured data.

Keywords: automata, logic, regular tree languages

1 Introduction

Regular tree languages can be defined by means of many equivalent formalisms, for instance: (non)deterministic bottom-up and nondeterministic top-down tree automata, alternating tree automata, two-way tree automata, homomorphic images of local tree languages, and monadic second-order logic [13,20]. However, it is not known whether there exists a natural inherently sequential model for recognizing the regular tree languages. Of course, by definition, they are recognized by bottom-up finite tree automata, but these automata are essentially parallel rather than sequential: the control of the automata is at several nodes of the input tree simultaneously, rather than at just one. With this aim in mind, Engelfriet, together with his co-workers Bloem, Hoogeboom, and van Best, initiated a research program [5,8,10] studying (extensions of) the tree-walking automata (TWAs) originally introduced by Aho and Ullman [2]. The finite control of a tree-walking automaton is always at one node of the input tree. Based on the label of that node and its child number (which is i if it is the ith child of its parent), the automaton changes state and steps to one of the neighboring nodes (parent or child). Without the test on the child number such automata cannot even search the tree in a systematic way, such as by a pre-order traversal as

⋆ Research Assistant of the Fund for Scientific Research, Flanders.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 547–560, 2000.
© Springer-Verlag Berlin Heidelberg 2000

is shown by Kamimura and Slutzki [14]. However, also with the child test, it is conjectured that these automata cannot express all regular tree languages [8,10]. In this paper, we study the expressiveness of tree-walking automata by characterizing them in terms of transitive closure logic formulas in normal form and prove the above mentioned conjecture for a restricted, but powerful, class of tree-walking automata.

Apart from the above purely theoretical motivation, recently, new interest in tree-walking automata emerged from the field of database theory. Indeed, one of the major research topics at the moment is the design and study of query languages for the manipulation of XML documents or electronic documents in general [1,16]. Such documents are usually modeled by ordered labeled trees or graphs, depending on the application at hand. In this research, tree-walking automata are used for various purposes and appeared in various forms. Milo, Suciu, and Vianu [15], for instance, used a transducer model based on tree-walking automata as a formal model for an XML transformer encompassing most current XML transformation languages. Brüggeman-Kleinn, Hermann, and Wood [6], proposed to use *caterpillar expressions* as a pattern language for XML transformation languages. Interestingly, caterpillar expressions relate to tree-walking automata like regular expressions relate to string automata: they are just a different, though a lot more user friendly, representation of the same thing. Furthermore, they conjectured their formalims to be less expressive than the regular tree languages. Another, more direct, occurrence of tree-walking automata is embodied in the actual XML transformation language XSLT [7] proposed by the World Wide Web consortium (W3C) and currently being implemented by IBM. In formal language theoretic terms, this query language can be best described as a *tree-walking tree transducer* [4]. Hence, results on the expressiveness of tree-walking automata could give insight in the expressiveness of actual XML transformation languages.

We start by characterizing the expressiveness of (deterministic and nondeterministic) tree-walking automata in terms of *(deterministic and non-deterministic) transitive closure logic* (DTC and TC) formulas in normal form. That is, formulas of the form $[(\mathrm{D})\mathrm{TC}(\varphi)](\varepsilon, \varepsilon)$, where φ is an FO formula containing predicates $\mathrm{depth}_m(x)$ defining x as a vertex whose depth is a multiple of m; and where ε refers to the root of the tree under consideration. Our result thus implies that any lower bound on (D)TC formulas in normal form is also a lower bound for (non)deterministic tree-walking automata. Unfortunately, proving lower bounds for the latter logic does not seem much easier than the original problem as Ehrenfeucht games for DTC and TC are quite involved. Therefore, we use a direct approach for a restricted, but expressive, class of tree-walking automata in the hope that these techniques will provide insight for the general case.

We first show that 1-bounded tree-walking automata, that is tree-walking automata that are only allowed to traverse every edge of the input tree at most once in every direction, cannot define all regular languages. In particular, we obtain that they can not evaluate tree-structured Majority circuits where the gates have fan-in greater than 2. Next, we generalize this result to a rather

powerful class of tree-walking automata, called r-restricted. These automata are rather expressive as they can define all of first-order logic (FO) and are capable of expressing some tree languages not definable in FO extended with regular path expressions. The latter logic is an abstraction of current query languages for semi-structured data and XML [1,16], and, for instance, cannot define the set of trees representing Boolean circuits evaluating to true [17] which can easily be defined by r-restricted tree-walking automata (cf. Example 1).

We conclude by mentioning some related work. Bargury and Makowsky [3] proved an equivalence between transitive closure logic and two-way multihead automata operating on grids. Their simulation of automata involves nesting of TC operators. Potthoff [18] showed that the same normal form of TC we use, suffices to define all regular *string* languages, the opposite direction being trivial in the string case. Recently, Engelfriet and Hoogeboom [9] showed that tree-walking automata with pebbles correspond exactly to TC. Hence, when allowing pebbles one can simulate nested TC operators. Fülöp and Maneth [11] showed that the domains of partial attributed tree transducers correspond to the tree-walking automata in universal acceptance mode.

This article is structured as follows. In Section 2, we define tree-walking automata. In Section 3, we prove the logical characterization of tree-walking automata in terms of transitive closure logic, and in Section 4 we proof the Engelfriet and Hoogeboom conjecture for two restrictions of tree-walking automata.

2 Preliminaries

Trees. A *tree domain* τ *over* $\mathbb{N}$ is a subset of $\mathbb{N}^*$, such that if $v \cdot i \in \tau$, where $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $v \in \tau$. Here, $\mathbb{N}$ denotes the set of natural numbers. If $i > 1$ then also $v \cdot (i-1) \in \tau$. The empty sequence, denoted by ε, represents the root. We call the elements of τ *vertices*. A vertex w is a *child* of a vertex v (and v the *parent* of w) if $vi = w$, for some i. A Σ*-tree* is a pair $t = (\mathrm{dom}(t), \mathrm{lab}_t)$, where $\mathrm{dom}(t)$ is a tree domain over $\mathbb{N}$, and lab_t is a function from $\mathrm{dom}(t)$ to Σ. The *arity* of a tree is the maximum number of children of its vertices. We only consider trees with a fixed arity. The depth of a vertex u, denoted by $\mathrm{depth}(u)$ is the length of u (interpreted as a string over $\mathbb{N}^*$). The *distance* between two vertices u and v, denoted by $d(u, v)$ is defined as the length of the path between u and v where we assume that $d(u, u) = 0$.

A Σ-tree t can be naturally viewed as a finite structure over the binary relation symbols E and $<$, and the unary relation symbols $(O_\sigma)_{\sigma \in \Sigma}$. E is the edge relation and equals the set of pairs $(v, v \cdot i)$ for every $v, v \cdot i \in \mathrm{dom}(t)$. The relation $<$ specifies the ordering of the children of a node, and equals the set of pairs $(v \cdot i, v \cdot j)$, where $i < j$ and $v \cdot j \in \mathrm{dom}(t)$. For each σ, O_σ is the set of nodes that are labeled with a σ.

Tree-Walking Automata. Tree-walking automata (TWAs) can be seen as the simplest analogon of two-way string automata. A TWA starts its computation in an initial state at the root of the input tree. In each step, it moves to a neighbor

vertex of the current vertex (or stays at the current vertex) and enters a state. The direction of movement and the new state depend only on the current state, the symbol at the current vertex, the child number of the current vertex, i.e., the relative position of the current vertex in the ordered list of the children of its parent, and the number of children of the current vertex.

More formally, a (k-ary) TWA is a tuple $(S, \Sigma, \delta, s_0, F)$, where S is the set of states, Σ is an alphabet (the set of possible vertex labels), $s_0 \in S$ is the initial state and $F \subseteq S$ is the set of accepting states. The only part where a TWA is formally different from a standard string automaton is the transition function δ. The transition function δ of a TWA is the union of the functions δ^i and $\delta^{\text{root},i}$, where $i \in \{0, \dots, k\}$. For a deterministic TWA,

- $\delta^{\text{root},i}$ is a function from $S \times \Sigma$ to $\{\text{stay}, \downarrow_1, \dots, \downarrow_i\} \times S$, and
- for each $i \in \{0, \dots, k\}$, δ^i is a function from $\{1, \dots, k\} \times S \times \Sigma$ to $\{\uparrow, \text{stay}, \downarrow_1, \dots, \downarrow_i\} \times S$.

For a nondeterministic TWA the ranges of these functions are the respective power sets. If several TWAs are around we write δ_M to denote the transition function of TWA M and the like.

A configuration $c = [v, s]$ of a TWA M on a tree t consists of a vertex v of t and a state s of M. The immediate successor configuration c' of a configuration $[v, s]$ is defined as follows.

- If $v = \epsilon$, v has i children and carries the symbol σ then
  - $c' = [\epsilon, s']$, if $\delta^{\text{root},i}(s, \sigma) = (\text{stay}, s')$, and
  - $c' = [j, s']$, if $\delta^{\text{root},i}(s, \sigma) = (\downarrow_j, s')$.
- If $v = wj$, for some $j \leq k$, v has i children and carries the symbol σ then
  - $c' = [w, s']$, if $\delta^i(j, s, \sigma) = (\uparrow, s')$,
  - $c' = [v, s']$, if $\delta^i(j, s, \sigma) = (\text{stay}, s')$, and
  - $c' = [vj', s']$, if $\delta^i(j, s, \sigma) = (\downarrow_{j'}, s')$.

We write $c \Rightarrow_{M,t} c'$ to express that c' is the immediate successor configuration of c in the computation of M on t. Following standard convention we write $c \Rightarrow^j_{M,t} c'$ ($c \Rightarrow^*_{M,t} c'$) to express that c' is the j-th (some) successor configuration of c in the computation of M on t. If M and/or t are clear from the context we may omit them. A tree t is *accepted* by M if there is an $s \in F$ such that $[\varepsilon, s_0] \Rightarrow^*_{M,t} [\varepsilon, s]$.

Example 1. We illustrate the above definition by means of an example. In particular we define a deterministic tree-walking automaton that accepts all tree-structured Boolean circuits of fan-in 2 that evaluate to true. A similar construction can be given for each fixed bound on the fan-in. For convenience, we only consider circuits of the right format. That is, all inner nodes and the root have exactly two children and are labeled with AND or OR. Further, all leaves are labeled by 0 or 1. These circuits are assigned a truth value in the usual way. Define M as the tuple $(S, \Sigma, \delta, \text{eval}, F)$ with $S = \{\text{eval}, 0, 1, \text{left-child-0}, \text{left-child-1}\}$, $\Sigma = \{\text{AND}, \text{OR}, 0, 1\}$, and $F = \{1\}$. The transition function δ is defined as follows:

– M starts by walking to the first leaf in the tree: for each $\sigma \in \{\text{AND}, \text{OR}\}$ and $i \in \{1, 2\}$,

$$\begin{aligned} \delta^{\text{root},2}(\text{eval}, \sigma) &= (\downarrow_1, \text{eval}); \text{ and} \\ \delta^2(i, \text{eval}, \sigma) &= (\downarrow_1, \text{eval}). \end{aligned}$$

– When reaching a zero (one) leaf, which is a left child, M moves up with this information:

$$\begin{aligned} \delta^0(1, \text{eval}, 0) &= (\uparrow, \text{left-child-0}); \\ \delta^0(1, \text{eval}, 1) &= (\uparrow, \text{left-child-1}); \end{aligned}$$

– If M enters an inner vertex from a left child it is always in one of the two states left-child-0 or left-child-1. If the current vertex is an AND vertex and the state is left-child-0 then the current vertex will evaluate to 0 no matter the right subtree. An analogous statement holds for OR-gates and the state left-child-1. The information passed to the parent vertex depends on whether the current vertex is itself a left or a right child. If the outcome of the left child is not sufficient to determine the value of the current vertex M has to enter its right child. Formally, for each $i \in \{1, 2\}$:

$$\begin{aligned} \delta^2(1, \text{left-child-0}, \text{AND}) &= (\uparrow, \text{left-child-0}); \\ \delta^2(2, \text{left-child-0}, \text{AND}) &= (\uparrow, 0); \\ \delta^2(1, \text{left-child-1}, \text{OR}) &= (\uparrow, \text{left-child-1}); \\ \delta^2(2, \text{left-child-1}, \text{OR}) &= (\uparrow, 1); \\ \delta^2(i, \text{left-child-1}, \text{AND}) &= (\downarrow_2, \text{eval}); \text{ and} \\ \delta^2(i, \text{left-child-0}, \text{OR}) &= (\downarrow_2, \text{eval}). \end{aligned}$$

– If M enters an inner vertex from a right child it is always in one of the states 0 or 1. By what we have said before, this state indicates the value of the subtree at the current vertex. Hence, it only has to be passed to its parent vertex. Consequently, for each $i \in \{0, 1\}$ and $\sigma \in \{\text{AND}, \text{OR}\}$:

$$\begin{aligned} \delta^2(1, 0, \sigma) &= (\uparrow, \text{left-child-0}); \\ \delta^2(1, 1, \sigma) &= (\uparrow, \text{left-child-1}); \text{ and} \\ \delta^2(2, i, \sigma) &= (\uparrow, i). \end{aligned}$$

– It remains to handle the case of leaves that are right children of their parent. They simply have to pass their value to the parent. For each $i \in \{0, 1\}$:

$$\delta^0(2, \text{eval}, i) = (\uparrow, i).$$

It will be convenient to subsume the overall effect of a TWA on a subtree of a tree in a so-called behaviour function. Intuitively, if f is the behaviour function of a subtree t then $f(s) = s'$ if and only if in the computation of M which starts at the root v of t in state s the parent of v is entered in state s'. Obviously, the behaviour function of a subtree t depends on the child number of its root in the full tree.

We define behaviour functions more formally. Let M be a k-ary deterministic TWA and let t be a (at most) k-ary tree. For $i \leq k$, let $t(i)$ denote the tree which consists of a root which has the root of t as the i-th child and $i-1$ other children which are leaves. The behaviour function $f_{M,t,i}$ of M on t as an i-th child maps states of M to states of M. It is defined as follows. If s is a state of M then $f_{M,t,i}(s) = s'$, if there is a j such that $(i, s) \Rightarrow^j_{M,t(i)} (\epsilon, s')$ and j is minimal with this property.

Note that $f_{M,t,i}$ does not depend on the labels of the vertices outside of t. Furthermore it does not depend either on the actual embedding of t in a larger tree as long as the root of t is an i-th child.

For non-deterministic TWAs behaviour functions are defined analogously but with sets of states as function values.

Next, we turn to the definition of some classes of restricted TWAs.

- We call a TWA 1-*bounded*, if, for all trees t, it traverses each edge of t at most once in each direction.
- We call a TWA M *r-restricted* if the following holds. For each pair u, v of vertices of a tree t such that $d(u, v) > r$ the computation of M on t does not contain 4 configurations $[u, s_1], [v, s_2], [u, s_3], [v, s_4]$ in the given order. Intuitively, this means that each path of length more than r is traversed at most once in each direction.

Clearly, each 1-bounded TWA is also r-restricted for every $r \geq 1$. Engelfriet and Hoogeboom showed that the latter automata can define all of first-order logic (FO) [8]. Moreover, r-restricted TWAs can even define some tree languages not definable in FO extended with regular path expressions [17]. In Section 4, we show that r-restricted TWAs cannot define the set of all regular tree languages thereby giving an answer to the conjecture of Engelfriet and Hoogeboom for a powerful class of TWAs.

Let, for $m > 0$, depth_m be a unary relation symbol. In the following we will consider trees which have additionally the predicate depth_m, for some m. In all trees, depth_m will contain all vertices the depth of which is a multiple of m. For a vertex u of a tree t, its r-sphere $S^t_r(u)$ is the set $\{v \mid d(u, v) \leq r\}$. For a tuple of vertices $\bar{u}$, define $S^t_r(\bar{u})$ as $\bigcup_{u \in \bar{u}} S^t_r(u)$. We define its r-neighborhood $N^t_r(\bar{u})$ as the structure t extended with the constants $\bar{u}$ restricted to the set $S^t_r(\bar{u})$.

3 A Logical Characterization of Tree-Walking Automata

We characterize tree-walking automata by *transitive closure logic formulas* of the form $\text{TC}[\varphi(x, y)](\varepsilon, \varepsilon)$, where φ is an FO formula, which may make use of the predicate depth_m, for some m. We say that such a formula is in *normal form*. Further,

$$t \models \text{TC}[\varphi(x, y)](\varepsilon, \varepsilon)$$

iff the pair $(\varepsilon, \varepsilon)$ is in the transitive closure of the relation $\{(u, v) \mid t \models \varphi[u, v]\}$. We use *deterministic transitive closure logic formulas* (DTC) in an analogously

defined normal form to capture deterministic tree-walking automata. In particular,

$$t \models \mathrm{DTC}[\varphi(x,y)](\varepsilon,\varepsilon)$$

iff the pair $(\varepsilon,\varepsilon)$ is in the transitive closure of the relation $\{(u,v) \mid t \models \varphi[u,v] \wedge (\forall z)(\varphi[u,z] \rightarrow z = v)\}$. The latter expresses that we disregard vertices u that have multiple φ-successors.

Theorem 2. *Nondeterministic tree-walking automata accept precisely the tree languages definable by TC formulas in normal form. Deterministic tree-walking automata accept precisely the tree languages definable by DTC formulas in normal form.*

Proof. (sketch) The simulation of TWA's by (D)TC formulas in normal form is an easy extension of a proof of Potthoff [18] who characterized two-way *string* automata by means of TC formulas in normal form. We omit the details.

For the other direction, consider the formula $\mathrm{TC}[\varphi(x,y)](\varepsilon,\varepsilon)$. By Gaifman's Theorem (see, e.g., [12]), there exists an r such that φ is equivalent to a Boolean combination of sentences χ and r-local formulas $\xi(x,y)$. Here, a formula $\xi(x,y)$ is r-local if for every tree t with vertices u and v, whether $t \models \xi[u,v]$ only depends on the isomorphism type of $N_r^t(u,v)$. As the rank of trees is fixed, there are only finitely many possible isomorphism types. Engelfriet and Hoogeboom [8] showed that TWA's can evaluate FO formulas over trees. A straightforward adaptation of their proof shows that TWA's can also evaluate FO formulas with modulo depth predicates (details omitted). Hence, we can assume that M can evaluate the sentences χ at the beginning of the computation. Further, if $d(u,v) \leq 2r$ then M can check whether $t \models \varphi[u,v]$ by inspecting $N_{3r}^t(u)$, otherwise by first inspecting $N_r^t(u)$ and afterwards $N_r^t(v)$. Suppose the automaton arrives at a vertex u (with $u = \varepsilon$ as the first case). First, the automaton nondeterministically chooses whether it will go to a vertex v of distance $\leq 2r$ or to a vertex v of distance $> 2r$. In the first case, it inspects $N_{3r}^t(u)$ and chooses a v such that $t \models \varphi[u,v]$ if this is possible. Otherwise it first computes the type of $N_r^t(u)$, moves to a vertex v of distance $> 2r$ and checks that the type of $N_r^t(v)$ implies that $t \models \varphi[u,v]$. Finally, if v is the root, the automaton accepts. Otherwise, it proceeds in the same manner. Clearly, the automaton will eventually accept if $t \models \mathrm{TC}[\varphi(x,y)](\varepsilon,\varepsilon)$.

The simulation of DTC formulas by deterministic TWAs is more intricate. Consider the formula $\mathrm{DTC}[\varphi(x,y)](\varepsilon,\varepsilon)$. We construct a deterministic TWA over k-ary trees accepting exactly the k-ary trees the above formula defines. As in the previous case, the automaton first evaluates all sentences χ in the Gaifman normal form of φ. Let m be the maximum number of vertices occurring in an r-neighborhood of a tree. That is, $m := \max\{|S_r^t(u)| \mid t \text{ a tree}, u \in \mathrm{dom}(t)\}$. For each isomorphism type τ of r-neighborhoods with one distinguished vertex, the automaton additionally computes the number of occurrences of τ in t up to $m+2$. Now, given a u, to find a v such that $\varphi(u,v)$ holds, the automaton proceeds as follows. Let Y_0 be the set of all vertices w in $N_{2r}^t(u)$ such that

$N_r^t(u,w) \models \varphi(x,y)$. By inspecting the $3r$-neighborhood of u, M can compute $|Y_0|$.

Next, it computes the type of $N_r^t(u)$ and the set T of all types τ of r-neighborhoods for which the following holds: if $N_r^t(w)$ is of type τ and $N_r^t(w)$ and $N_r^t(u)$ are disjoint then $N_r^t(u,w) \models \varphi(x,y)$. The latter is a fixed finite computation which can be encoded into the transition function. We call vertices of a type from T *good* vertices (w.r.t. the current u) and denote the set of good vertices in t by Y_1. Note that M can deduce $|Y_1|$ (up to $m+2$) from the precomputed information.

The set of good vertices in $N_{2r}^t(u)$ is denoted by Y_2. By inspecting the $3r$-neighborhood of u again, M computes $|Y_2|$ and the relative positions of all vertices in Y_2 w.r.t. u.

Let $Y = Y_0 \cup (Y_1 - Y_2)$. Y is the set of vertices w of t such that $N_r^t(u,w) \models \varphi(x,y)$. M can compute $|Y|$ without further moving (note that $|Y_2| \leq m$). If $|Y|$ is different from 1 then M can immediately reject. Let us assume in the following that $|Y| = 1$.

If the unique element v of Y is from Y_0, M can directly go to v. If, on the other hand, $Y_0 = Y_2 = \emptyset$ then M can move to the unique $v \in Y_1$ via a DFS traversal of the tree.

The only complicated case is when $Y_0 = \emptyset$ and $|Y_1 - Y_2| = 1$ but $Y_2 \neq \emptyset$. The complication arises from the following possibility. M has to traverse the tree to find the correct unique good w outside $N_{2r}^t(u)$. As it does not know in which part of the tree w is located its way to w might lead back to u. In that case we have to make sure that it does not confuse w with a vertex from Y_2.

We can assume w.l.o.g. that the root of the tree is not in $N_{3r}^t(u)$ because otherwise M can easily distinguish the vertices of Y_2 from the desired vertex. Now, M proceeds as follows. It starts a DFS walk through the tree starting from u and first inspecting the first subtree of u. Whenever it encounters a new vertex z it starts a subcomputation which inspects the $3r$-neighborhood of z to find out whether there is a good vertex w in $N_{2r}^t(z)$. If such aw is found then M computes the set $Z(w)$ of vertices $u' \in N_{2r}^t(w)$ for which $N_{3r}^t(u') \cong N_{3r}^t(u)$. If $Z(w) = \emptyset$ or all vertices in $Z(w)$ are behind z in the DFS order then w is the desired vertex v and M goes there. Otherwise it proceeds in its DFS walk. When the DFS walk finishes at the root (without finding the target vertex) then M walks back (reverse DFS) until it reaches u again (easily recognized by the isomorphism type of its $3r$-neighbourhood). Then it starts a reverse DFS walk from u (going upwards first) analogously to the first DFS walk.

We have to show that M always finds the correct target vertex. First, we show that M never moves to a vertex in Y_2. Let $w \in Y_2$. If M reaches w during the inspection of the neighborhood of a vertex z before its DFS walk arrives at the root then $u \in Z(w)$ and M recognizes that u is in the DFS order before z. Hence it does not take w as v. The analogous statement is true if z is found in the reverse DFS walk after coming back to u.

Finally, we show that M indeed reaches a target vertex v from Y_1 (which then is the correct one). Assume wlog that v is encountered as a vertex w relative

to a vertex z in the DFS walk behind u. If $Z(v) = \emptyset$ then v is easily identified. Assume $Z(v) \neq \emptyset$. As v is not in $N^t_{2r}(u)$ all vertices in $Z(v)$ are behind u in the DFS order. Let u' be the first vertex from $Z(v)$ in the DFS order. Then v is found not later than at the time u' is assumed as z in the DFS walk.

4 Weakness of Tree-Walking Automata

Let k be fixed and let T_k be the set of all k-ary trees the leaves of which are labeled with 0 or 1. For each vertex v of a tree $v \in T_k$ we inductively define a value 0 or 1 as follows. If v is a leaf then its value (0 or 1) is determined by its label (0 or 1). If v has i children then v has the value 1 if and only if at least $\frac{i}{2}$ of its children have the value 1. Intuitively, T_k is the set of all tree-structured Majority circuits. Let T^1_k (T^0_k) denote the set of all trees $t \in T_k$ for which the root gets the value 1 (0). We call a vertex v of a tree $t \in T_k$ a *1-vertex* (*0-vertex*) if it gets the value 1 (0). Analogously, we call the subtree rooted at a 1-vertex (0-vertex) a *1-subtree* (*0-subtree*). The next lemma follows immediately from the inductive definition of the set T^k.

Lemma 3. *For each k, the set T^1_k is a regular set of trees.*

We have seen in Example 1 that there is a deterministic TWA which recognizes T^1_2 (note that $T^1_2 \cup T^0_2$ can be seen as the set of trees representing Boolean circuits consisting of only OR-gates). This was due to the fact that the automaton, after evaluating a right subtree of a vertex v, could conclude the value of the corresponding left subtree of v from the label of v and the fact that it had to enter the right subtree. For $k > 2$, things are more complicated. In fact, we conjecture the following.

Conjecture 4. For $k > 2$, T^1_k can not be recognized by a (det. or nondet.) TWA.

In this section we prove this conjecture for a restricted type of TWAs, 1-bounded TWAs. The proof can be easily generalized to the sets T^1_k, for each $k > 3$. Furthermore, it can be extended to show that, for each r there is a related regular set of trees which can not be recognized by any r-restricted TWA.

Before we state and prove the result we introduce some important concepts for that proof and show a purely combinatorial result. For any $d \geq 1$ a *critical tree* of depth d is a full ternary tree t of depth d with the following properties: (i) $t \in T^1_3$; (ii) each 1-vertex of t has exactly two children which are 1-vertices; and (iii) each 0-vertex has only 0-vertices as children. In particular, there are no 1-leaves in 0-subtrees of t. Intuitively, a critical tree of depth d is a tree of depth d from T_3 which contains a full binary subtree of depth d which only has 1-leaves and all other leaves are 0-leaves. In particular, a critical tree of depth d has 2^d 1-leaves. A *numbering* N of a critical tree t of depth d is an injective mapping of the 1-leaves of t into the set $\{0, \ldots, 2^d - 1\}$. All critical trees of a fixed size d are defined on the same tree domain τ_d. A mapping M which maps each leaf of τ_d to a subset of $\{0, \ldots, 2^d - 1\}$ with at most m elements is called an *m-labeling*. We say that an m-labeling M of τ_d is *compatible* with a numbering

N of a critical tree t of depth d if, for each 1-leaf v of t, $N(v) \in M(v)$. We state the following lemma without proof.

Lemma 5. *For each $m > 0$ there is a $d > 0$ such that, for each m-labeling M of τ_d there is a critical tree of depth d for which there is no numbering that is compatible with M.*

We are now ready to prove the main result of this paper.

Theorem 6. *(a) There is no 1-bounded (deterministic or non-deterministic) TWA which recognizes T_3^1.*
(b) For each $r > 0$, there is is a regular tree language that can not be recognized by an r-restricted (deterministic or non-deterministic) TWA.

Proof. (sketch) The main task is to prove (a). Statement (b) will follow by an easy generalization of that proof. Towards a contradiction assume there exists a non-deterministic TWA M' which recognizes T_3^1. The proof consists of 3 main steps:

- We transform M' into a TWA M which accepts exactly the same trees as M' but always rejects when it visits a 0-leaf. From this property we can conclude that if M accepts a certain tree t then it also accepts every tree which results from t by replacing 0-subtrees with 1-subtrees.
- We show that there are two trees $t, t' \in T_3^1$ with the same tree domain such that M has accepting computations for these trees which enter some vertex v in the same state s but with different "histories". Here, the history consists of the information for which vertices w on the path from v to the root the computation visits the (only) 1-sibling of w before it visits v. This part of the proof makes use of Lemma 5.
- Finally, we show that these accepting computations can be combined into one accepting computation on a tree from T_3^0.

Before we describe the construction of M, we introduce some notation. Let, for $i \in \{1, \dots, 3\}$, $F^1_{M',i}$ ($F^0_{M',i}$) denote the set $\{f_{M',t,i} \mid t \in T_3^1\}$ ($\{f_{M',t,i} \mid t \in T_3^0\}$) of behaviour functions that M' can have on 1-subtrees (0-subtrees) that have child number i. A straightforward argument shows that M' can never recognize the set T_3^1 when $F^1_{M',i} \cap F^0_{M',i} \neq \emptyset$ for some i. Therefore, we can assume that $F^1_{M',i}$ and $F^0_{M',i}$ are disjoint for all i.

We turn to the construction of M. Intuitively, the idea is as follows. Whenever M' can go from a vertex v to vi then M can either do the same or, to prevent visiting a 0-leaf, it can guess that vi is the root of a 0-subtree. In the latter case instead of going down to vi it picks a behaviour function $f \in F^0_{M',i}$ and enters a new state at v according to f. Formally, for each $j \leq k$, $s \in S$ and $\sigma \in \Sigma - \{0\}$,

$$\delta^i_M(j, s, \sigma) = \delta^i_{M'}(j, s, \sigma) \cup \bigcup_{\substack{(\downarrow_{j'}, s') \in \delta^i_{M'}(j,s,\sigma) \\ f \in F^0_{M',j'}}} (\text{stay}, f(s')).$$

For each $j \leq k$ and $s \in S$, we define $\delta^i_M(j, s, 0) = (\text{stay}, \perp)$ where $\perp$ is a state from which no transition is possible.

We have to show that M accepts exactly all trees in T^1_3. Let therefore t be from T^1_3, hence, by assumption, t is accepted by M'.

Let $C' = c'_0, c'_1, \ldots, c'_n$ be an accepting computation of M' on t. We show that there is also an accepting computation $C = c_0, \ldots, c_m$ of M on t. We construct C by suitably modifying C'. Let $v = wj$ be a 0-leaf of t, for some vertex w and some j. By the prerequisite on M', v is at most visited once in C'. If v is not visited at all, we do not need to modify C' with respect to v. If v is visited then there are successive configurations $[w, s_i], [v, s_{i+1}], [w, s_{i+2}]$ in C'. Here, we assume w.l.o.g. that M' moves in each step. In C we replace these 3 configurations by $[w, s_i], [w, s_{i+2}]$. This reflects a legal transition of M as it corresponds to the computation of M' on the 0-subtree which consists of a single 0-leaf. We end up with a legal accepting computation C on t.

For the opposite direction, let $C = c_0, \ldots, c_m$ be an accepting computation of M on a tree $t \in T_3$. We construct a tree t' by replacing some of the 0-leaves of t by 0-trees and an accepting computation C' of M' on t' by modifying C accordingly. By the construction of t' it will follow that $t \in T^1_3$ if and only if $t' \in T^1_3$. Hence, as M' accepts t' it also accepts t.

More formally, let $[v, s], [v, s']$ be two successive configurations from C such that it does not hold $[v, s] \Rightarrow_{M',t} [v, s']$. Hence, this subcomputation is possible only by the new transitions introduced in M. By definition, there must be $j \leq k$, $s'' \in S$, $f \in F^0_{M',j}$ and a 0-tree t_0 such that $[v, s] \Rightarrow_{M',t} [vj, s'']$ and $f_{M',t_0,j}(s'') = s'$. From this we can conclude that there exist configurations $c(v, 1), \ldots, c(v, l)$ such that $[v, s], [vj, s''], c(v, l), \ldots, c(v, l), [v, s']$ is a legal subcomputation of M' on the tree, denoted by $t(v)$, in which the subtree rooted at vj is replaced by the 0-subtree t_0. If $t \in T^0_3$ then also $t(v) \in T^0_3$ as we only replaced a (0- or 1-) subtree by a 0-subtree. By inductively applying this argument we arrive at a tree t' and an accepting computation C' on t'. Hence, by assumption, $t' \in T^1_3$ and therefore $t \in T^1_3$.

It should be noted that the latter construction relies on the assumption that M' traverses each edge at most once. Otherwise, it could be the case that the configuration C visits v two times but the extension to C' makes use of two different 0-subtrees rooted at v.

Let now t be a critical tree of depth d and let $C = c_0, \ldots, c_n$ be an accepting computation of M on t. If C does not visit all 1-leaves then we can easily construct a tree $t' \in T^0_3$ which is accepted by M. Hence, we assume that C visits each 1-leaf of t exactly once. Let v be such a 1-leaf of t and let $c_j = [v, s]$ be the configuration of C which visits it. Let, for $i \in \{1, \ldots, d\}$, v_i denote the vertex of depth i on the path from the root of t to v. As v is a 1-leaf, each v_i is a 1-vertex. Therefore, as t is critical, each vertex v_i has exactly one sibling v'_i which is a 1-vertex. For each i, v'_i is visited in exactly one of the subcomputations $c_0, \ldots, c_{j-1}$ and $c_{j+1}, \ldots, c_m$. We define, for each 1-leaf v its *history string* $z = h_{t,C}(v) = z_1 \cdots z_d$ by setting $z_i = 1$ if and only if v'_i is visited in $c_0, \ldots, c_{j-1}$. These history strings have a couple of nice properties which are straightforward to prove given the assumptions on M.

- The binary number $b_{t,C}(v)$ represented by $z = h_{t,C}(v)$ (where z_d is the least significant bit) coincides with the number of 1-leaves that are visited in C before v. This follows from the restriction that the automaton can visit each subtree at most once. Hence, when a 1-vertex v at level i is entered, then all 2^{d-i} 1-leaves in the subtree of v have to be visited before the subtree is left. In this way, a 1 at the i-th bit of the history string corresponds to 2^{d-i} 1-leaves that have been already visited.
- Consequently, the function $b_{t,C}$ defines a numbering of t. In particular, each 0-1-string occurs exactly once as a history string of a 1-leaf v.

We claim that, there exists a d, two critical trees t, t' of depth d, two accepting computations C, C' of M on t, t', respectively, and a leaf v of τ_d such that (i) v is a 1-leaf of t and t'; (ii) C and C' visit v in the same state s; (iii) and, $h_{C,t}(v) \neq h_{C',t'}(v)$.

Towards a contradiction assume that this claim is false. Let m be the number of states of M and let d be a number as given by Lemma 5. Let t and t' be two critical trees of depth d, let v be a common 1-leaf of t and t' and let C and C' be accepting computations of t and t', respectively, which visit v in the same state s. The assumption implies that $h_{C,t}(v) = h_{C',t'}(v)$ and therefore $b_{C,t}(v) = b_{C',t'}(v)$. We can conclude that, for each vertex v of τ_d and each state s of M, there is only one number $n(v, s)$ such that $b_{C,t}(v) = n(v, s)$ for all critical trees t with 1-leaf v and all accepting computations which visit v in the state s. In other terms, there exists an m-labeling M of the leaves of τ_d such that, for each critical t and each accepting computation C on t the numbering $b_{C,t}$ is compatible with M. This contradicts Lemma 5, as desired. Therefore, the claim is proved.

Let d, t, t', C, C' and v be as given by the above claim. We complete the proof by constructing a tree $t_0 \in T_3^0$ which is accepted by M. Let $z = h_{C,t}(v)$ and $z' = h_{C',t'}(v)$. Let j be minimal such that $z_j \neq z'_j$. We can assume w.l.o.g. that $z_j = 0$ and $z'_j = 1$. Let, for each $i \in \{1, \dots, d\}$, v_i be defined as above, w_i be the 1-sibling of v_i in t and w'_i be the 1-sibling of v_i in t'. We construct t_0 as follows: (i) for each $i < j$, if $z_i = 1$ (i.e., w_i is visited before v in C) then we copy the subtrees rooted at the siblings of v_i from t; (ii) for each $i < j$, if $z_i = 0$ (i.e., w'_i is visited before v in C') then we copy the subtrees rooted at the siblings of v_i from t' (iii) at the siblings of v_j we root 0-subtrees (this assures that t_0 is a 0-tree); and, (iv) in the subtree rooted at v_j all leaves are labeled 1.

Let $C = c_0, \dots, c_m$, $C' = c'_0, \dots, c'_n$ and let k and k' be such that $c_k = c'_{k'}$ are the configurations in which v is visited.

It is straightforward to check that

- $c_0, \dots, c_k$ is a valid subcomputation on t_0 because all 1-leaves of t that are visited in $c_0, \dots, c_k$ are also 1-leaves in t_0;
- $c'_{k'}, \dots, c'_n$ is a valid subcomputation on t_0 because all 1-leaves of t' that are visited in $c'_{k'}, \dots, c'_n$ are also 1-leaves in t_0; hence
- $c_0, \dots, c_k = c'_{k'}, \dots, c'_n$ is an accepting computation on t_0, the desired contradiction.

This concludes the proof of statement (a).

To prove (b) we use a slightly different set U_3^1 of trees. These trees have an additional label, +. Inner vertices that are labeled with + have 3 children which are interpreted as threshold gates. Inner vertices that are not labeled with + have only 1 child. Hence, at a +-vertex there are starting 3 paths which lead either to another +-vertex or to a leaf. Now a +-vertex is evaluated to 1, if at least 2 of its 3 descendants (+-vertex or leaf) evaluate to 1. Intuitively, U_3^1 is the same as T_3^1 but the edges of trees in T_3^1 are replaced by paths in U_3^1. In fact, the proof of the fact that no r-restricted TWA recognizes U_3^1 is almost word for word the proof given in (a) but in the trees that are used, each edge has to be replaced by a path of length $r+1$. The old vertices are labeled with +, the new ones not.

Acknowledgements

We thank Joost Engelfriet, Martin Grohe, Hendrik Jan Hoogeboom, and Clemens Lautemann.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.
2. A. V. Aho and J. D. Ullman. Translations on a context-free grammar. *Inform. and Control*, 19:439–475, 1971.
3. Y. Bargury and J. A. Makowsky. The expressive power of transitive closure logic and 2-way multi-head automata. In E. Borger, G. Jäger, H. K. Büning, and M. M. Richter editors. *Computer Science Logic*, volume 626 of *Lecture Notes in Computer Science*, pages 1–14. Springer-Verlag, 1991.
4. G. Bex, S. Maneth, and F. Neven. A formal model for an expressive fragment of XSLT. Submitted.
5. R. Bloem and J. Engelfriet. A comparison of tree transductions defined by monadic second order logic and by attribute grammars. To appear in the *Journal of Computer and System Sciences.*
6. A. Brüggeman-Klein, S. Hermann, and D. Wood. Context, caterpillars, tree automata, and tree pattern matching. In Proceedings of the *4th International Conference on Developments in Language Theory*, 1999.
7. J. Clark. XSL Transformations version 1.0. http://www.w3.org/TR/xslt.
8. J. Engelfriet and H. J. Hoogeboom. Tree-walking pebble automata. In J. Karhumäki, H. Maurer, G. Paun, and G.Rozenberg, editors, *Jewels are forever, contributions to Theoretical Computer Science in honor of Arto Salomaa*, pages 72–83. Springer-Verlag, 1999.
9. J. Engelfriet and H. J. Hoogeboom. Private communication.
10. J. Engelfriet, H.J. Hoogeboom, and J.-P. van Best. Trips on trees. *Acta Cybernetica*, 14:51–64, 1999.
11. Z. Fülöp and S. Maneth. Domains of partial attributed tree transducers. *Information Processing Letters*, 175–180, 2000.
12. H. Gaifman. On local and nonlocal properties. In J. Stern, editor, *Logic Colloquium '81*, pages 105–135. North Holland, 1982.
13. F. Gécseg and M. Steinby. Tree languages. In [19], chapter 1.

14. T. Kamimura and G. Slutzki. Parallel and two-way automata on directed ordered acyclic graphs. *Information and Control*, 49(1):10–51, April 1981.
15. T. Milo, D. Suciu, and V. Vianu. Type checking for XML transformers. Proceedings of the *19th ACM Symposium on Principles of Database Systems*, 2000.
16. F. Neven. *Design and Analysis of Query Languages for Structured Documents — A Formal and Logical Approach.* Doctor's thesis, Limburgs Universitair Centrum, 1999.
17. F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. Proceedings of the *19th ACM Symposium on Principles of Database Systems*, 2000.
18. A. Potthoff. Logische Klassifizierung regulärer Baumsprachen. Doctor's thesis, Institut für Informatik u. Prakt. Math., Universität Kiel, 1994.
19. G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Languages.* Springer, 1997.
20. W. Thomas. Languages, automata, and logic. In [19], chapter 7.

Determinization of Transducers over Infinite Words

Marie-Pierre Béal[1] and Olivier Carton[2]

[1] Institut Gaspard Monge,
Université de Marne-la-Vallée
Marie-Pierre.Beal@univ-mlv.fr
http://www-igm.univ-mlv.fr/~beal/
[2] Institut Gaspard Monge and CNRS
Université de Marne-la-Vallée
Olivier.Carton@univ-mlv.fr
http://www-igm.univ-mlv.fr/~carton/

Abstract. We study the determinization of transducers over infinite words. We consider transducers with all their states final. We give an effective characterization of sequential functions over infinite words. We also describe an algorithm to determinize transducers over infinite words.

1 Introduction

The aim of this paper is the study of determinization of transducers over infinite words, that is of machines realizing rational transductions. A transducer is a finite state automaton (or a finite state machine) whose edges are labeled by pairs of words taken in finite alphabets. The first component of each pair is called the input label. The second one the output label. The rational relation defined by a transducer is the set of pairs of words which are labels of an accepting path in the transducer. We assume that the relations defined by our transducers are functions which each string of the domain to a string. This is a decidable property [8].

The study of transducers has many applications. Transducers are used to model coding schemes (compression schemes, convolutional coding schemes, coding schemes for constrained channels, for instance). They are widely used in computer arithmetic [7] and in natural language processing [13]. Transducers are also used in programs analysis [6]. The determinization of a transducer is the construction of another transducer which defines the same function and has a deterministic (or right resolving) input automaton. Such transducers allow a sequential encoding and thus are called sequential transducers.

The characterization of sequential functions on finite words was obtained by Choffrut [4,5]. His proof contains implicitly an algorithm for determinization of a transducer. This algorithm has also been described by Mohri [11] and Roche and Shabes [13, p. 223–233]. In this paper, we address the same problem for infinite words. We consider transducers and functions over infinite words and

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 561–570, 2000.
© Springer-Verlag Berlin Heidelberg 2000

our transducers have all their states final. The reason why we assume that all states are final is that the case of transducers with final states seems to be much more complex. Indeed, the determinization of automata over infinite words is already very difficult [14]. In particular, it is not true that any rational set of infinite words is recognized by a deterministic automaton with final states and Büchi acceptance condition. Other accepting conditions, as the Muller condition for instance, must be used.

We first give an effective characterization of sequential functions over infinite words. This characterization extends to infinite words the twinning property introduced by Choffrut [4]. We prove that a function is sequential if it is a continuous map whose domain can be recognized by a deterministic Büchi automaton, and such that the transducer obtained after removing some special states has the twinning property. These conditions can be simplified in the case where the transducer has no cycling path with an empty output label. We use this characterization to describe an algorithm checking whether a function realized by a transducer is sequential. This algorithm becomes polynomial when the transducer has no cycling path with an empty output label. Finally, we give an algorithm to determinize a transducer. The algorithm is much more complex than in the case of finite words.

The paper is organized as follows. Section 2 is devoted to basic notions of transducers and rational functions. We give in Sect. 3 a characterization of sequential functions while the algorithm for determinization of transducers is described in Sect. 4.

2 Transducers

In the sequel, A and B denote finite alphabets. The set of finite and right-infinite words over A are respectively denoted by A^* and A^ω. The empty word is denoted by ε. The set A^ω is endowed with the usual topology induced by the following metric: the distance $d(x, y)$ is equal 2^{-n} where n is the minimum $\min\{i \mid x_i \neq y_i\}$. In this paper, a function from A^ω to B^ω is said to be continuous iff it is continuous with respect to this topology.

A *transducer* over $A \times B$ is composed of a finite set Q of *states*, a set $E \subseteq Q \times A^* \times B^* \times Q$ of *edges* and a set $I \subseteq Q$ of *initial* states. An edge $e = (p, u, v, q)$ from p to q is denoted by $p \xrightarrow{u|v} q$. The words u and v are called the *input label* and the *output label*. Thus, a transducer is the same object as an automaton, except that the labels of the edges are pairs of words instead of letters. In the literature, transducers also have a set of final states. In this paper, we only consider transducers all of which states are final and with Büchi acceptance condition. Any infinite path which starts at an initial state is then successful. We omit the set of final states in the notation.

An infinite *path* in the transducer $\mathcal{A}$ is an infinite sequence

$$q_0 \xrightarrow{u_0|v_0} q_1 \xrightarrow{u_1|v_1} q_2 \xrightarrow{u_2|v_2} q_3 \cdots$$

of consecutive edges. Its input label is the word $x = u_0u_1u_2\ldots$ whereas its output label is the word $y = v_0v_1v_2\ldots$. The path is said to *start* at q_0.

An infinite path is then *successful* if it starts at an initial state. A pair (x, y) of infinite words is *recognized* by the transducer if it labels a successful path. A transducer defines then a relation $R \subseteq A^\omega \times B^\omega$. The transducer computes a function if for any word $x \in A^\omega$, there exists at most one word $y \in B^\omega$ such that $(x, y) \in R$. We call it the function *realized* by the transducer. Thus a transducer can be seen as a machine computing nondeterministically output words from input words. We denote by $\mathrm{dom}(f)$ the domain of the function f. A transducer that realizes a function can be transformed in an effective way in a transducer labelled in $A \times B^*$ that realizes the same function. These transducers are sometimes called *real time transducers*.

Let $\mathcal{T}$ be a transducer. The *underlying input automaton* of $\mathcal{T}$ is obtained by omitting the output label of each edge. A transducer $\mathcal{T}$ is said to be *sequential* if it is labeled in $A \times B^*$ and if the following conditions are satisfied.

- it has a unique initial state,
- the underlying input automaton is deterministic.

These conditions ensure that for each word $x \in A^\omega$, there is at most one word $y \in B^\omega$ such that (x, y) is recognized by $\mathcal{T}$. Thus, the relation computed by $\mathcal{T}$ is a partial function from A^ω into B^ω. A function is *sequential* if it can be realized by a sequential transducer.

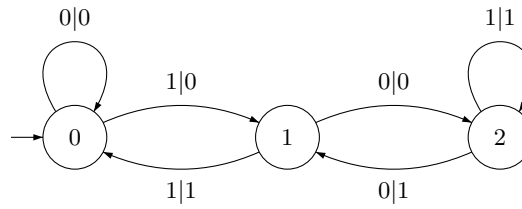


Fig. 1. Transducer of Example 1

Example 1. Let $A = \{0, 1\}$ be the binary alphabet. Consider the sequential transducer $\mathcal{T}$ pictured in Fig. 1. If the infinite word x is the binary expansion of a real number $\alpha \in [0, 1)$, the output corresponding to x in $\mathcal{T}$ is the binary expansion of $\alpha/3$. The transducer $\mathcal{T}$ realizes the division by 3 on binary expansions. The transducer obtained by exchanging the input and output labels of each edge realizes of course the multiplication by 3. However, this new transducer is not sequential.

3 Characterization of Sequential Functions

In this section, we characterize functions realized by transducers with all states final that can be realized by sequential transducers. This characterization uses topological properties of the function and some twinning property of the transducer. It extends the result of Choffrut [4,5] to infinite words.

The characterization of the sequentiality is essentially based on the following notion introduced by Choffrut [5, p. 133] (see also [3, p. 128]). Two states q and q' of a transducer are said to be *twinned* if and only if for any pair of paths

$$i \xrightarrow{u|u'} q \xrightarrow{v|v'} q$$
$$i' \xrightarrow{u|u''} q' \xrightarrow{v|v''} q',$$

where i and i' are two initial states, the output labels satisfy the following property. Either $v' = v'' = \varepsilon$ or there exists a finite word w such that either $u'' = u'w$ and $wv'' = v'w$, or $u' = u''w$ and $wv' = v''w$. The latter case is equivalent to the following two conditions:

(i) $|v'| = |v''|$,
(ii) $u'v'^{\omega} = u''v''^{\omega}$

A transducer has the *twinning property* if any two states are twinned.

Before stating the main result, we define a subset of states which play a particular role in the sequel. We say that a state q of a transducer is *constant* if all infinite paths starting at this state have the same output label. This unique output is an ultimately periodic word. It should be noticed that any state accessible from a constant state is also constant. We now state the characterization of sequential functions.

Proposition 1. *Let f be a function realized by a transducer $\mathcal{T}$. Let $\mathcal{T}'$ be the transducer obtained by removing from $\mathcal{T}$ all states which are constant. Then the function f is sequential if and only if the following three properties hold:*

- *the domain of f can be recognized by a deterministic Büchi automaton,*
- *the function f is continuous,*
- *the transducer $\mathcal{T}'$ has the twinning property.*

Since the function f is realized by a transducer, the domain of f is rational. However, it is not true that any rational set of infinite words is recognized by a deterministic Büchi automaton. Landweber's theorem states that a set of infinite words is recognized by a deterministic Büchi automaton if and only if it is rational and G_δ [16]. Recall that a set is said to be G_δ is it is equal to a countable union of open sets for the usual topology of A^ω.

It is worth pointing out that the domain of a function realized by a transducer may be any rational set although it is supposed that all states of the transducer are final. The final states of a Büchi automaton can be encoded in the outputs of a transducer in the following way. Let $\mathcal{A} = (Q, E, I, F)$ be a Büchi automaton.

We construct a transducer $\mathcal{T}$ by adding an output to any transition of $\mathcal{A}$. A transition $p \xrightarrow{a} q$ of $\mathcal{A}$ becomes $p \xrightarrow{a|v} q$ in $\mathcal{T}$ where v is empty if p is not final and is equal to a fixed letter b if p is final. It is clear that the output of a path is infinite if and only if the path goes infinitely often through a final state. Thus the domain of the transducer $\mathcal{T}$ is the set recognized by $\mathcal{A}$. For instance, the domain of a transducer may be not recognizable by a deterministic Büchi automaton as in the following example. It is however true that the domain is closed if the transducer has no cycling path with an empty output.

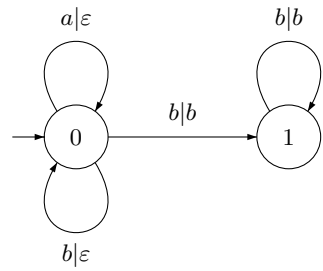


Fig. 2. Transducer of Example 2

Example 2. The domain of the function f realized by the transducer of Fig. 2 is the set $(a+b)^*b^\omega$ of words having a finite number of a. The function f cannot be realized by a sequential transducer since its domain is not a G_δ set.

It must be also pointed out that a function realized by a transducer may be not continuous although it is supposed that all states of the transducer are final as it is shown in the following example.

Example 3. The image of an infinite word x by the function f realized by the transducer of Fig. 3 is $f(x) = a^\omega$ if x has an infinite number of a and $f(x) = a^n b^\omega$ if the number of a in x is n. The function f is not continuous. For instance, the sequence $x_n = b^n a b^\omega$ converges to b^ω while $f(x_n) = ab^\omega$ does not converge to $f(b^\omega) = b^\omega$.

Before describing the algorithm for determinization, we first study a particular case. It turns out that the first two conditions of the proposition are due to the fact that the transducer $\mathcal{T}$ may have cycling paths with an empty output. If the transducer $\mathcal{T}$ has no cycling path with an empty output, the previous proposition can be stated in the following way.

Proposition 2. *Let f be a function realized by a transducer $\mathcal{T}$ which has no cycling path with an empty output. Let $\mathcal{T}'$ be the transducer obtained by removing*

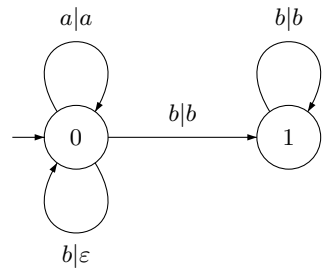


Fig. 3. Transducer of Example 3

from $\mathcal{T}$ all states which are constant. Then the function f is sequential if and only if the transducer $\mathcal{T}'$ has the twinning property.

The previous proposition can be directly deduced from Proposition 1 as follows. If the transducer $\mathcal{T}$ has no cycling path with an empty output, any infinite path has an infinite output. Thus, an infinite word x belongs to the domain of f if and only if it is the input label of an infinite path in $\mathcal{T}$. The domain of f is then a closed set. It is then recognized by a deterministic Büchi automaton whose all states are final. This automaton can be obtained by the usual subset construction on the input automaton of $\mathcal{T}$. Furthermore, if the transducer $\mathcal{T}$ has no cycling path with an empty output, the function f is necessarily continuous.

We now study the decidability of the conditions of Propositions 1 and 2. We have the following results.

Proposition 3. *It is decidable if a function f given by a transducer with all states final is sequential. Furthermore, if the transducer has no cycling path with an empty output, this can be decided in polynomial time.*

A Büchi automaton recognizing the domain of the function can be easily deduced from the transducer. It is then decidable if this set can be recognized by a deterministic Büchi automaton [16, thm 5.3]. However, this decision problem is NP-complete.

It is decidable in polynomial time whether a function given by a transducer with final states is continuous [12]. The twinning property of a transducer is decidable in polynomial time [2].

4 Determinization of Transducers

In this section, we describe an algorithm to determinize a transducer which satisfies the properties of Proposition 1. This algorithm proves that the conditions of the proposition are sufficient. The algorithm is exponential in the number of states of the transducer.

Let $\mathcal{T} = (Q, E, I)$ be a transducer labelled in $A \times B^*$ that realizes a function f. Let $\mathcal{T}'$ be the transducer obtained by removing from $\mathcal{T}$ all states which are constant. We assume that $\mathcal{T}'$ has the twinning property. We denote by C the set of states which are constant. For a state q of C, we denote by y_q, the unique output of q which is an ultimately periodic word. We suppose that the domain of f is recognized by the deterministic Büchi automaton $\mathcal{A}$. This automaton is used in the constructed transducer to ensure that the output is infinite only when the input belongs to the domain of the function.

We describe the deterministic transducer $\mathcal{D}$ realizing the function f. Roughly speaking, this transducer is the synchronized product of the automaton $\mathcal{A}$ of the domain and of an automaton obtained by a variant of the subset construction applied on the transducer. In the usual subset contruction, a state of the deterministic automaton is a subset of states which memorizes all accessible states. In our variant of the subset construction, a state is a subset of pairs formed of a state and a word which is either finite of infinite.

A state of $\mathcal{D}$ is a pair (p, P) where p is a state of $\mathcal{A}$ and P is a set containing two kinds of pairs. The first kind are pairs (q, z) where q belong to $Q \setminus C$ and z is a finite word over B. The second kind are pairs (q, z) where q belongs to C and z is an ultimately periodic infinite word over B. We now describe the transitions of $\mathcal{D}$. Let (p, P) be a state of $\mathcal{D}$ and let a be a letter. Let R be equal to the set defined as follows

$$\begin{aligned} R = &\{(q', zw) \mid q' \notin C \text{ and } \exists (q, z) \in P,\ q \notin C \text{ and } q \xrightarrow{a|w} q' \in E\} \\ &\cup \{(q', zwy_{q'}) \mid q' \in C \text{ and } \exists (q, z) \in P,\ q \notin C \text{ and } q \xrightarrow{a|w} q' \in E\} \\ &\cup \{(q', z) \mid q' \in C \text{ and } \exists (q, z) \in P,\ q \in C \text{ and } q \xrightarrow{a|w} q' \in E\} \end{aligned}$$

We now define the transition from the state (p, P) input labeled by a. If R is empty, there is no transition from (p, P) input labeled by a. Otherwise, the output of this transition is the word v defined as follows. Let $p \xrightarrow{a} p'$ be the transition in $\mathcal{A}$ from p labeled by a. If p' is not a final state of $\mathcal{A}$, we define v as the empty word. If p' is a final state, we define v as the first letter of the words z if R only contains pairs (q', z) with $q' \in C$ and if all the infinite words z are equal. Otherwise, we define v as the longest common prefix of all the finite or infinite words z for $(q', z) \in R$. The state P' is then defined as follows

$$P' = \{(q', z) \mid (q', vz) \in R\}$$

There is then a transition $(p, P) \xrightarrow{a|v} (p', P')$ in $\mathcal{D}$. The initial state of $\mathcal{D}$ is the pair $(i_{\mathcal{A}}, J)$ where $i_{\mathcal{A}}$ is the initial state of $\mathcal{A}$ and where $J = \{(i, \varepsilon) \mid i \in I \text{ and } i \notin C\} \cup \{(i, y_i) \mid i \in I \text{ and } i \in C\}$. If the state p' is not final in $\mathcal{A}$, the output of the transition from (p, P) to (p', P') is empty and the words z of the pairs (q, z) in P, may have a nonempty common prefix. We only keep in $\mathcal{D}$ the accessible part from the initial state. The transducer $\mathcal{D}$ has a deterministic input automaton. It turns out that the transducer $\mathcal{D}$ has a finite number of states.

The following proposition finally states that the sequential transducer $\mathcal{D}$ is finite and that it is equivalent to the transducer $\mathcal{T}$. Both transducers realize the same function over infinite words.

Proposition 4. *The sequential transducer $\mathcal{D}$ has a finite number of states and it realizes the same function f as the transducer $\mathcal{T}$.*

It is not straightforward that the transducer $\mathcal{D}$ has actually a finite number of states. It must be proved that the finite words which occur as second component of the pairs in the states are bounded. It follows then that the infinite words occuring as second component of the pairs are suffixes of a finite number of ultimately periodic words. Therefore, there are finitely many such words.

It must also be proved that the transducer $\mathcal{D}$ realizes the same function as $\mathcal{T}$. This follows mainly from the following lemma which states the key property of the edges in $\mathcal{D}$.

Lemma 1. *Let u be a finite word. Let $(i_{\mathcal{A}}, J) \xrightarrow{u|v} (p, P)$ be the unique path in $\mathcal{D}$ with input label u from the initial state. Then, the state p is the unique state of $\mathcal{A}$ such that $i_{\mathcal{A}} \xrightarrow{u} p$ is a path in $\mathcal{A}$ and the set P is equal to*

$$P = \{(q, z) \mid \exists\, i \xrightarrow{u|v'} q \text{ in } \mathcal{T} \text{ such that } v' = vz \text{ if } q \notin C \\ v'y_q = vz \text{ if } q \in C\}$$

This construction is illustrated by the following example.

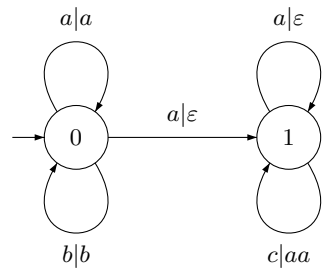


Fig. 4. Transducer of Example 4

Example 4. Consider the transducer pictured in Fig. 4. A deterministic Büchi automaton recognizing the domain is pictured in Fig. 5. If the algorithm for determinization is applied to this transducer, one gets the transducer pictured in Fig. 6.

These determinizations do not preserve the dynamic properties of the transducers as the locality of its output automaton. Recall that a finite automaton is *local* if any two biinfinite paths with the same label are equal. We mention that in [9], an algorithm is given to determinize transducers over bi-infinite words

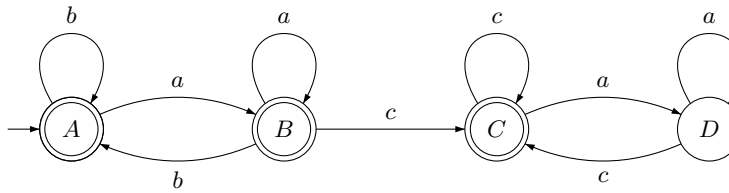


Fig. 5. A deterministic Büchi automaton for the domain

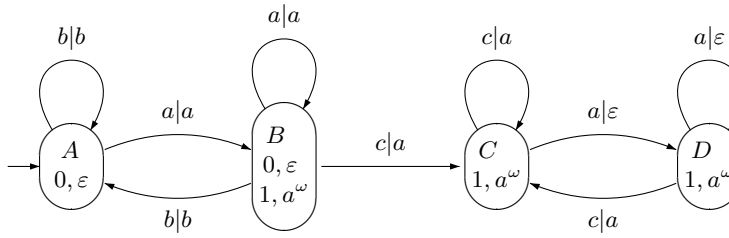


Fig. 6. Determinization of the transducer of Fig. 4

that have a right closing input (or that are n-deterministic or deterministic with a finite delay in the input) and a local output (see also [10, p. 143] and [1, p. 110–115]). This algorithm preserves the locality of the output. These features are important for coding applications.

Acknowledgments

The authors would like to thank Jean Berstel for very helpful suggestions and Christian Choffrut and Isabelle Fagnot for their relevant comments on a preliminary version of this paper.

References

1. Marie-Pierre Béal. *Codage Symbolique.* Masson, 1993.
2. Marie-Pierre Béal, Olivier Carton, Chrhistophe Prieur, and Jacques Sakarovitch. Squaring transducers. In *LATIN'2000.*
3. Jean Berstel. *Transductions and Context-Free Languages.* B.G. Teubner, 1979.
4. Christian Choffrut. Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles. *Theoret. Comput. Sci.*, 5:325–338, 1977.
5. Christian Choffrut. *Contribution à l'étude de quelques familles remarquables de fonctions rationnelles.* Thèse d'État, Université Paris VII, 1978.
6. A. Cohen and J.-F. Collard. Instance-wise reaching definition analysis for recursive programs using context-free transductions. In *PACT'98*, 1998.

7. Christiane Frougny. Numeration systems. In M. Lothaire, editor, *Algebraic Combinatorics on Words*. Cambridge, 1999. to appear.
8. F. Gire. Two decidability problems for infinite words. *Inform. Proc. Letters*, 22:135–140, 1986.
9. Bruce Kitchens. *Continuity properties of factor maps in ergodic theory*. Ph.D. thesis, University of North Carolina, Chapel Hill, 1981.
10. Doug Lind and Brian Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.
11. Mehryar Mohri. On some applications of finite-state automata theory to natural languages processing. *Journal of Natural Language Engineering*, 2:1–20, 1996.
12. Christophe Prieur. How to decide continuity of rational functions on infinite words. *Theoret. Comput. Sci.*, 1999.
13. Emmanuel Roche and Yves Schabes. *Finite-State Language Processing*, chapter 7. MIT Press, Cambridge, 1997.
14. Shmuel Safra. On the complexity of ω-automata. In *29th Annual Symposium on Foundations of Computer Sciences*, pages 24–29, 1988.
15. Ludwig Staiger. Sequential mappings of ω-languages. *RAIRO-Infor. Théor. et Appl.*, 21(2):147–173, 1987.
16. Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 133–191. Elsevier, 1990.

Constraint Programming and Graph Algorithms

Kurt Mehlhorn

Max-Planck-Institut für Informatik
Im Stadtwald
66123 Saarbrücken
Germany
http://www.mpi-sb.mpg.de/~mehlhorn

In the spring of '99 my colleague Gert Smolka gave me a short introduction to constraint programming. During our discussion Gert emphasized that the search for efficient propagation algorithms leads to hard and well motivated questions in algorithmics. He pointed me to the papers [Reg94] by J.-C. Regin on "A filtering algorithm for constraints of difference in CSPs" and [GC97,GC00] by N. B. Guernalec and A. Colmerauer on "Narrowing a 2n-block of sortings in $O(n \log n)$". I soon learned that Gert had pointed me to an extremely rich source of algorithmic problems which I am now exploring in cooperation with E. Althaus and S. Thiel from my research group, D. Duchier and J. Niehren from the programming systems lab, A. Koller from the computer linguistics department at the Universität des Saarlandes, and with Nicolas Beldiceanu at SICS. Some papers [MT00,ADK+00,KMN00] and implementations of propagators for the Oz system (http://www.ps.uni-sb.de/oz2/) have come out of the cooperation so far.

The purpose of this talk is to get more researchers from the algorithms community interested in the subject. I want to make the case that constraint programming offers a lot of very challenging algorithmic problems and that cooperations between the constraint programming and the algorithms community could be very beneficial to both communities.

1 Constraint Programming

I give a brief account of constraint programming based on [Smo96]. For more thorough discussions we refer the reader to [Smo96,MS98]. Examples of constraint programming systems are ILOG, Chip, Eclipse, and Oz.

Constraint programming is a powerful programming paradigm, that allows one to formulate computational problems at a very high level. A computational problem is formulated as a *constraint*. A constraint is simply a conjunction of formulae of first-order logic. A solution is an assignment of values to the (free) variables which satisfies the constraint. It is the task of the constraint programming system to find a (all, a best) solution. The programmer only needs to specify the problem[1]. Readers entirely unfamiliar with constraint programming

[1] In most constraint programming systems, the programmer can also specify the search strategy, but this is not important at the current level of discussion.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 571–575, 2000.
© Springer-Verlag Berlin Heidelberg 2000

should move on to the next section and have a look at the specification of the N-queens problem given there.

Two types of constraints are distinguished: *basic* and *non-basic* constraints. This distinction is more a pragmatic distinction than a mathematical distinction. Basic constraints are constraints that can be handled efficiently and non-basic constraints are difficult constraints.

The basic constraints must have two properties:

- There is an efficient procedure which, given a collection of basic constraints, decides their satisfiability, or exhibits a solution, or exhibits all solutions.
- Fixing the value of a variable turns a set of basic constraints into a set of basic constraints.

We come to non-basic constraints. Non-basic constraints are difficult. They are simplified by means of iterated branching and propagation steps. A branching step splits the current state into two: the first is obtained by adding some constraint and the second is obtained by adding its negation. Branching generates a tree of states. Propagation operates on the current leaves of this tree and tries to simplify the states associated with them.

A *propagator* is associated with every non-basic constraint C. Let B denote the conjunction of all basic constraints. The invocation of a propagator has one of the following effects.

- The propagator may declare C obsolete. If C is declared obsolete, B must entail C, i.e., any assignment satisfying B must also satisfy C.
- The propagator may declare inconsistency. In this case $B \wedge C$ must be unsatisfiable.
- The propagator *advances to* D. Here D is a basic constraint that is entailed by $B \wedge C$ and that is strictly stronger than B, i.e., D is not entailed by B.
- The propagator cannot make any progress[2].

The computation in a leaf can stop when there is no further propagator or when a propagator declares inconsistency. In the former situation B defines the set of solutions. In the latter case, the leaf contributes no solution.

When the computation has not stopped yet and no propagator can make any progress, a distribution step (branching step) is taken. We invent a constraint C and proceed from the unsolved space to two new spaces, the first obtained by adding a propagator for C and the second obtained by adding a propagator for the negation $\neg C$.

2 An Example: The Alldiff Constraint

We use the alldiff constraint to illustrate the connection between propagation and graph algorithms.

[2] Propagators must have some minimal capabilities. In particular, propagators must make progress if B determines a single assignment. In this case a propagator for C must either declare C obsolete or must declare inconsistency.

Consider the following situation. We have a set of variables $x_1, \ldots, x_n$. The basic constraints define for each variable x_i a finite set V_i of conceivable values. We also want the values of the variables to be pairwise distinct (alldiff constraint). The well known N-queens problem is a toy example which shows the usefulness of the alldiff constraint.

The goal is to place n queens on a $n \times n$ checker board. There can be at most one queen in each row, column, diagonal, and anti-diagonal. Assume that we place the i-th queen in position (i, x_i) (this definition makes sense since we have exactly one queen per row), $1 \leq i \leq n$. Every queen excludes a diagonal (45° line) and a anti-diagonal (−45° line) If we identify a diagonal and anti-diagonal with the square which it uses in row 0, then a queen in position (i, x_i) uses the diagonal $y_i = i + x_i$ and the anti-diagonal $z_i = i - x_i$. Thus we have the constraints:

$$x_i \in [1 \,..\, n], \quad y_i \in [2 \,..\, 2n], \quad z_i \in [-n+1 \,..\, n-1],$$
$$y_i = x_i + i, \quad z_i = i - x_i,$$
$$\mathit{Alldifferent}(x_1, \ldots, x_n), \quad \mathit{Alldifferent}(y_1, \ldots, y_n), \quad \mathit{Alldifferent}(z_1, \ldots, z_n)$$

The first line defines the conceivable values for our variables, the second line defines relations between pairs of variables, and the third line defines three alldifferent constraints. Propagators for the equalities in the second line are easily designed. If a value v becomes impossible for x_i, the value $v + i$ becomes impossible for y_i, and vice versa.

Regin [Reg94] observed that matching theory leads to efficient propagators for the alldiff constraint. He suggested to view an alldiff constraint as a matching problem in a bipartite graph. On the left side of the bipartite graph there is a node for each variable and on the right side of the graph there is a node for each conceivable value. A variable x is connected to a value v if v is a possible for x.

An alldiff constraint is satisfiable iff the graph G just defined has a matching in which all variables are matched (a variable-perfect matching). This is a well studied problem in algorithmics. There is a solution to the alldiff constraint in which variable x has value v iff there is variable-perfect matching containing the edge (x, v). This is also a question which has been discussed in the literature on matching. Puget gave the following answer.

Let M be any matching involving all variables. We orient G as follows: Direct all edges in M from right to left and all edges not in M from left to right. An edge $(x, v) \notin M$ is part of a perfect matching if there is an alternating cycle containing it or an alternating path starting in a free node on the right side and ending in a matched node on the right hand side. In the directed version of G, the first kind of edge is an edge in any strongly connected component and the second kind of edge is any edge that is reachable from a free node on the right hand side. Thus given a matching, narrowing takes linear time. If no matching is available, narrowing takes time $O(\sqrt{n}m)$. Isn't this a nice application of matching theory?

In the two preceding sentences we talked about narrowing instead of propagation because propagation for the alldiff constraint amounts to narrowing the sets of potential values of the variables.

We can go further. Suppose we have just narrowed an alldiff constraint and then perform branching, e.g., we may partition the set of values of some variable into two. This will define two new bipartite graphs. Of course, we would like to work on these graphs without explicitly constructing them. That's a typical problem in incremental graph algorithms.

Frequently, the conceivable values of a variable form an interval of natural numbers. The bipartite graph can then be represented in space $O(n)$ and we may be interested in narrowing algorithms which reduce the sizes of these intervals. The question arises whether this can be done without constructing the bipartite graph defined above. Puget [Pug98] answered this question positively and described an $O(n \log n)$ algorithm for obtaining bound consistency. The algorithm is again based on matching theory. In graph-theoretic terms the question amounts to computing perfect matchings and strongly components of so-called convex bipartite graph. In a convex bipartite graph, the nodes on the right side are linearly ordered and the neighbors of each node on the left side form an interval in the right side. The matching problem in bipartite convex graphs was first studied by [Glo67]. In [MT00] S. Thiel and I exploit the connection and obtain simplified and faster narrowing algorithms for the alldiff and the sortedness constraint.

3 Further Reading

In the previous section we tried to make the point that propagation rises interesting algorithmic questions. How can you learn more about these problems?

- The most effective method is probably to talk to a researcher in the constraint programming community.
- Read [VHS+97] on strategic directions in constraint programming.
- Check the proceedings of the conference on "Principles and Practice of Constraint Programming (CP)" [MP98,Jaf99].
- Check the journal "Constraints" (Kluwer Academic Publishers).
- Read the paper [Bel00] by Nicolas Beldiceanu. The paper classifies constraints using graph terminology.

References

ADK+00. E. Althaus, D. Duchier, A. Koller, K. Mehlhorn, J. Niehren, and S. Thiel. Efficient algorithms for dominance problems. `www.mpi-sb.mpg.de/~mehlhorn/ftp/dominance.ps`, March 2000.

Bel00. N. Beldiceanu. Global constraints a graph properties on structured networks of elementary constraints of the same type. Technical Report SICS T2000/01, SICS, Uppsala, Sweden, 2000.

GC97. N. B. Guernalec and A. Colmerauer. Narrowing a 2n-block of sortings in O(n log n). *Lecture Notes in Computer Science*, 1330:2–16, 1997.

GC00. N. B. Guernalec and A. Colmerauer. Optimal narrowing a block of sortings in optimal time. *Constraints*, 5:85–118, 2000.

Glo67. F. Glover. Maximum matchings in a convex bipartite graph. *Naval Res. Logist. Quart.*, 14:313–316, 1967.

Jaf99. Joxan Jaffar, editor. *Proceedings of the Fifth International Conference on Principles and Practice of Constraint Programming*, volume 1713 of *Lecture Notes in Computer Science*. Alexandra, VA, USA, October 1999.

KMN00. A. Koller, K. Mehlhorn, and J. Niehren. A polynomial-time fragment of dominance constraints. `www.coli.uni-sb.de/~koller/papers/poly-dom.ps.gz`, April 2000.

MP98. Michael Maher and Jean-François Puget, editors. *Proceedings of the Forth International Conference on Principles and Practice of Constraint Programming*, volume 1520 of *Lecture Notes in Computer Science*. Pisa, Italy, October 1998.

MS98. Kim Marriott and Peter J. Stuckey. *Programming with Constraints: An Introduction*. MIT Press, Cambridge, MA, 1998.

MT00. K. Mehlhorn and Sven Thiel. Faster algorithms for bound-consistency of the sortedness and the alldifferent constraint. `www.mpi-sb.mpg.de/~sthiel/sortedness_paper.ps.gz`, March 2000.

Pug98. Jean-François Puget. A fast algorithm for the bound consistency of alldiff constraints. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, pages 359–366, Menlo Park, July 26–30 1998. AAAI Press.

Reg94. J.-C. Regin. A filtering algorithm for constraints of difference in CSPs. In *Proc. 12th Conf. American Assoc. Artificial Intelligence*, volume 1, pages 362–367. Amer. Assoc. Artificial Intelligence, 1994.

Smo96. Gert Smolka. Problem solving with constraints and programming. *ACM Computing Surveys*, 28(4es), December 1996. Electronic Section.

VHS+97. Pascal Van Hentenryck, Vijay Saraswat, et al. Strategic directions in constraint programming. *Computing Surveys*, 28(4):701–726, December 1997. ACM 50th Anniversary Issue. Strategic Directions in Computing Research.

Scalable Secure Storage when Half the System Is Faulty

Noga Alon[1], Haim Kaplan[1], Michael Krivelevich[1], Dahlia Malkhi[2], and Julien Stern[3]

[1] School of Mathematical Sciences, Tel Aviv University, Israel
[2] School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel
[3] Laboratoire de Recherche en Informatique, CNRS - Universite de Paris Sud, France

Abstract. In this paper, we provide a method to safely store a document in perhaps the most challenging settings, a highly decentralized replicated storage system where up to half of the storage servers may incur arbitrary failures, including alterations to data stored in them.
Using an error correcting code (ECC), e.g., a Reed-Solomon code, one can take n pieces of a document, replace each piece with another piece of size larger by a factor of $\frac{n}{n-2t}$ such that it is possible to recover the original set even when up to t of the larger pieces are altered. For t close to $n/2$ the space overhead of this scheme is close to n, and an ECC such as the Reed-Solomon code degenerates to a trivial replication code.
We show a technique to reduce this large space overhead for high values of t. Our scheme blows up each piece by a factor slightly larger than two using an erasure code which makes it possible to recover the original set using $n/2 - O(n/d)$ of the pieces, where $d \approx 80$ is a fixed constant. Then we attach to each piece $O(d \log n / \log d)$ additional bits to make it possible to identify a large enough set of unmodified pieces, with negligible error probability, assuming that at least half the pieces are unmodified, and with low complexity. For values of t close to $n/2$ we achieve a large asymptotic space reduction over the best possible space blowup of any ECC in deterministic setting. Our approach makes use of a d-regular expander graph to compute the bits required for the identification of $n/2 - O(n/d)$ good pieces.

1 Introduction

In order to safeguard a document, the most simple solution is to replicate it, and to store the different copies in different places. This method, however, has two main drawbacks. First, the integrity of multiple replicas is harder to maintain, and second the required storage space grows linearly with the number of copies. In this paper, we provide a method to safely store a document that addresses both issues. First, our method guarantees integrity against arbitrary alterations, even malicious ones, in up to half of the storage servers. Second, the storage costs remain reasonable even in large systems, composed of hundreds or thousands of servers.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 576–587, 2000.
© Springer-Verlag Berlin Heidelberg 2000

1.1 Our Contribution

Our approach makes use of an erasure code (that can recover the information provided some pieces are lost but the ones that remain are required to be correct) and adds verification information to the code pieces. For our computations, we assume usage of IDA [Rab89] whose space blow-up is optimal, though other erasure codes, e.g. [LMS+97,AL96], may be employed for efficiency with a slight sacrifice in space overhead. Let n be the number of pieces. We arrange the pieces in a graph, called the *storage graph*, such that each piece is a vertex, and an edge exists between vertices when they cross verify each other. Each vertex stores fingerprint information that transitively verifies every vertex up to distance k away in the graph, where k is a parameter chosen at setup time. A fingerprint is a digest of fixed length representing the content of a piece. Fingerprints have the property that it is highly unlikely (and infeasible) to find two different pieces with the same fingerprint. Typically, a cryptographically secure hash function, e.g., SHA1 [SHA1], is used as a digest function. The transitive verification information takes only a factor k more space than a regular fingerprint of the adjacent pieces. Herein lies a large gain, since each vertex verifies a neighborhood in the graph of radius k, which grows exponentially. The total storage cost is $O(kdn)$, where d is a bound on the storage graph degree. When $kd \ll n$, this cost is a significant improvement over previous methods. The complexity of our recovery and storage algorithms is $O(kdn)$ in addition to the time required for decoding and encoding the erasure code of choice. Our algorithm needs to compute, in the worst case, only kdn digests. The range of parameters which will be of particular interest for us is when d is constant and k is $O(\log n)$.

The storage graph we employ has the property that even when up to $t < n/2$ of its vertices are removed, a sufficiently large component of size $\Theta(n)$ remains connected with diameter $\leq k$. For this, we make use of known constructions of expander graphs [LPS86] and prove that the required properties hold in them. That is, we prove that if up to $t < n/2$ vertices are removed from an expander like that in [LPS86], then there remains a component of size $n/2 - O(n/d)$ with diameter $O(\log n/\log d)$. This result is of independent interest and may have other applications. Furthermore it can be extended to a setup when more than half the vertices are removed.

The retrieval algorithm selects a vertex at random and collects all the vertices that are verified by it, by a simple breadth-first-search. We show that this selection procedure needs to be repeated only an expected constant number of times until it collects a linear set of correct vertices. The total number of fingerprinting computations is bounded by $O(dn \log n/\log d)$. The computation of fingerprints dominates the time overhead of our retrieval algorithm over the decoding complexity of the erasure code we use.

1.2 Related Work and Alternative Approaches

The most prevalent approach for achieving resilience to arbitrary server corruption is the state machine approach [Lam79,Sch90], which numerous systems

employ. Using this approach in the context of secure file storage, every server stores a full replica of the file and processes every update on it. The alteration of data stored by servers can be masked by obtaining $t + 1$ identical replicas, where t is presumed to be a bound on the total number of corrupted replicas. Unfortunately, this method has a high overhead in storing full copies of the file at each replica.

When alterations to stored data are not of concern, erasure codes solve the problem. For example, Rabin [Rab89] presents a solution, called Information Dispersal Algorithm (IDA), which allows to transform a document of size s into n pieces of size s/m, where m is a parameter chosen by the user, such that the document can be reconstructed from any m pieces. Since the total amount of space taken by m pieces is exactly s, the space overhead of IDA is clearly optimal. However, if any of the obtained pieces is altered, the integrity of the reconstructed document may be compromised. Moreover, a user obtaining such an erroneous document has no way of detecting that an error has occurred, and may simply return erroneous results undetectably.

To overcome this problem, it is necessary to add redundant information to pieces when they are stored, that indicates when some other pieces(s) are altered. A simple approach is to store a fingerprint of the entire document with each piece. To recover the file, first one gets the correct fingerprint from a majority of the pieces, and then checks combinations of pieces for a file with the same fingerprint. However, this may lead to prohibitive computations in searching for a right combination of unaltered pieces.

To obtain a feasible solution one could use an error correcting code (ECC). An ECC takes n pieces and blows up each piece with some additional information such that it is possible to recover all the pieces provided that $n - t$ are uncorrupted, for any $t < n/2$. The minimal space-blowup factor of any ECC is $n/(n - 2t)$ where n is the number of pieces. There are well known ECCs that achieve this optimal space overhead, e.g., Reed-Solomon codes. Unfortunately, when t approaches $n/2$, the space blows up by a factor of n (pieces) and this degenerates to simple replication.

Instead of using an ECC on the pieces themselves one can apply it to a shorter sequence of digests of the pieces, thereby reducing the space overhead at the expense of getting only a probabilistic guarantee for recovery. For example, the Secure IDA method in [Kra93] computes a *fingerprint* for each piece, and stores the vector of fingerprints using an ECC. To recover the document, first the vector of fingerprints is recovered, and then each piece is checked against its fingerprint. The space blow-up factor for storing a document with this method is $n/(n-t)$ for the IDA pieces, and an additional space for pieces of the fingerprints vector, blown up by a factor of $n/(n - 2t)$. Here, too, when t approaches $n/2$, the fingerprints vector is fully replicated. The space for the fingerprints vector depends only on n and the digest function used, and does not depend on the document length. Nevertheless, this space could be quite prohibitive when n is large. To illustrate this, suppose a file size is 1 Mega-Byte, fingerprints are 160

bits, $n = 1000$ and $t = 499$. Then Secure IDA stores roughly 1000×160 extra bits, or $\approx 20KBytes$, with every IDA piece of $1MB/(1000 - 499) \approx 2KBytes$.

We can reduce the large blowup factor of ECC (either on the pieces themselves or on the fingerprints) by using a list decoding algorithm. The general idea is to use an ECC which is able to correct less errors than the maximum possible. In case the number of errors is larger than what the ECC is capable of fixing the list decoding algorithm will generate a small list of possible decodings of which we will be able to choose the right one with high probability. Polynomial list decoding algorithms for Reed-Solomon codes have been recently discovered by Sudan [Su97]. More specifically, a Reed-Solomon code codes K blocks into N blocks, such that any two codewords differ in at least $N - K + 1$ blocks (the distance). If at most $(N - K)/2$ blocks are altered, there is a single codeword that is closest to the altered data (i.e., differs from it in fewer than $(N - K)/2$ different blocks). This is the highest error for which Reed-Solomon is guaranteed to retrieve the original document. As already mentioned, for this error rate to reach half the blocks, Reed-Solomon blows up a stored document by a factor close to its size, thus triviliazing to full replication.[1]

In case the number of errors is larger than half the code distance a Reed-Solomon code can be used to recover a *list* of all possible decodings. The number of possible decodings is constant as long as the number of errors is less than $N - \sqrt{NK}$ (see [GRS95]) and the problem of finding the list is known as the *list-decoding* problem. Using techniques introduced by Sudan [Su97] and subsequently improved in [RR00] and [GS99], such a list is produced with a randomized polynomial time algorithm. In its most efficient form [GS99], their scheme corrects up to $\lceil N - \sqrt{NK} - 1 \rceil$ errors. The scheme can be used to address our problem as follows: A document of length K blocks is encoded into $n = N = 4K$ blocks using a Reed-Solomon code. In addition, we store with each block a digest H of the full document. To retrieve the document when up to half of the blocks may be altered, we recover H from the majority, and employ Guruswami and Sudan's list-decoding method to obtain a list of possible decodings, which we compare against H to retrieve the original document with high probability. The space blow-up of this method is constant ($= 4$). The drawback of this scheme is the complexity of the retrieval algorithm which employs rather complicated methods, such as polynomial factorization, and has complexity cubic in n.[2] By comparison, our retrieval method is simpler (using only hashing and comparisons), and runs in $O(n \log(n))$ time. We use a completely different approach whose building blocks may have other applications.

A comparison of the efficiency of our method when half the system may be faulty with the various known approaches is given in Table 1 below.

[1] In previous paragraphs we thought of the document as decomposed into n pieces where n is fixed and the encoding increases the piece size. Here we think of the piece size as fixed and the encoding translates K pieces to N.

[2] The method by Roth and Ruckenstein [RR00] has computation complexity $O(n^2 \log^2 n)$ but needs twice as much space.

Table 1. Comparison of methods: Storing document of size s on n servers when up to $t = n/2$ may be faulty.

Method	Space overhead per server	Store time†	Retrieve time†
Our method	$(2+\epsilon)s/n + O(\log n)$	$O(n \log n)$	$O(n \log n)$
Simple replication	s	none	none
Reed-Solomon	s	none	none
Secure IDA	$2s/n + O(n)$	$O(n)$	$O(n)$
List decoding	$4s/n$	none	$O(n^3)$

† In addition to underlying coding/decoding time of the corresponding erasure or error correcting code.

Going back to our basic motivation, the need for scalable and survivable storage is reinforced in numerous recent systems that support information sharing in highly decentralized settings. Examples are the Eternity service [And96], a survivable digital document repository, SFS [MK98], a secure file system for a wide area network, Fleet [MR99], a survivable and scalable data replication system, a Byzantine file system of Castro et al. [CL99], and IBM's Evault [GGJ97], a storage system that employs Rabin's IDA to achieve survivable storage with reasonable storage burden. The verification information stored in these systems to guard against possible alteration of pieces does not scale to large system sizes. Our methods are most suitable for all the systems mentioned above and others, where scaling is a necessity.

The methods presented in this paper are concerned with the integrity of file storage and retrieval. Other aspects of data security are orthogonal to ours. Specifically, methods for preserving the secrecy of file contents in replicated systems have been proposed, e.g., in [HT88,AE90], such that the collusion of up to t faulty servers cannot reveal the contents of the information stored. These methods use secret sharing techniques that can be combined with our approach to achieve secrecy.

2 Preliminaries

The goal of this work is to provide two functions, *Share* and *Reconstruct*. Function *Share* takes a document x and produces n pieces denoted by $Share(x, 1), \ldots,$ $Share(x, n)$. Function *Reconstruct* recovers the document with high probability despite arbitrary alterations in up to a threshold $t = \lfloor \frac{n-1}{2} \rfloor$ of the pieces.

Our algorithms make use of a cryptographically secure hash function H (such as SHA1 [SHA1]). For any value v, in an unlimited range, $H(v)$ has fixed size (in bits). We assume that it is computationally infeasible to find two different values v and v' such that $H(v) = H(v')$. Typically, setting $|H|$ to 160 bits suffices to guarantee this today, e.g., with SHA1, and hence we will assume this.

We also use Rabin's IDA [Rab89]. At a high level, IDA takes a data-value x and converts it into n pieces, $IDA(x, 1), ..., IDA(x, n)$, such that recovery of

x is possible from any combination of m pieces, and such that the total space taken by every m components is precisely $|x|$ (optimal).

3 *Share*

The *Share* transformation takes a document x and produces n pieces $Share(x, 1)$, ..., $Share(x, n)$ to be stored on n corresponding servers. The goal of the *Share* transformation is to allow retrieval of the original document x despite arbitrary corruption of up to t of the pieces. To cope with such alterations, we will transform x into n pieces and store verification information on each piece in such a way that discrimination between correct and incorrect pieces can be achieved at a low cost, while maintaining a low storage overhead. The challenge is to minimize the storage requirements to enable our scheme to scale up to very large systems. The secrecy of the document is not the main concern, and can be added using standard methods.

Our solution first transforms x using IDA into n pieces, $IDA(x, 1)$, ..., $IDA(x, n)$, such that x can be restored from any subset of $(\frac{n}{2} - \epsilon n)$ pieces (ϵ will be determined in Section 3.1). To safeguard against alteration of pieces, we add to each piece verification information as follows. Pieces are arranged in a *store graph* $ST = (V, E)$ on n vertices (which will also be specified in Section 3.1). We denote the set of vertices adjacent to a vertex i in ST by $N(i)$. Each vertex in ST represents one piece, and it stores k levels of verification information. For every vertex i, we define level-ℓ verification information V_i^ℓ recursively as follows: $V_i^0 = IDA(x, i)$ and for $l \geq 1$

$V_i^\ell = \langle H(V_{j_1}^{\ell-1}), \ldots, H(V_{j_{|N(i)|}}^{\ell-1})\rangle$ where $j_1 < \ldots < j_{|N(i)|}$ are the neighbors of i.

In other words the level j verification information stored with piece i is the tuple of hashes of the level $j-1$ verification information stored at its neighbors. Each piece stores k levels of verification information that intuitively verify the pieces up to distance k away from i in the graph. (The parameter k will be determined in the next section). In addition, it stores the hash of the whole file.

The total space taken by each piece of a document x stored with our method is at most

$$|H|(dk + 1) + |x|/(\frac{n}{2} - \epsilon n) \ ,$$

where d is the maximum degree of any node in ST. When $dk = o(n)$, we get a significant improvement over ECCs. Note also that since the space overhead is proportional to the product of d and k, we can trade increased degree with decreased diameter, and vice versa. Hence, to be useful for storage and retrieval, we need the storage graph ST to have the following features:

- **Low degree**: The degree of vertex i in ST determines the storage overhead of the i'th piece.
- **Good expansion**: The expansion of ST determines the number of vertices that are at distance k from any particular vertex or group of vertices, and hence, the number of vertices that can be verified by them.

During retrieval, up to t vertices of ST may be corrupted, and hence, some set D of edges incident with $t \leq \lfloor \frac{n-1}{2} \rfloor$ vertices are removed. Notice that we do not know who are the t corrupt vertices and get to see a graph after deleting only the edges in D that are a subset of the edges incident with those t corrupt vertices. Nevertheless, using k-transitive verification, we know that every neighborhood of diameter k is either all correct or all corrupt. Hence, our graph construction needs to guarantee that after the removal of the set D of edges incident with t vertices from ST, there remains a set of $\frac{n}{2} - \epsilon n$ good vertices that are connected with a low diameter. We proceed to show such a construction.

3.1 Determining ST

We consider the problem of finding a storage graph ST such that when an arbitrary set D of edges incident with a set of $t = \lfloor \frac{n-1}{2} \rfloor$ malicious vertices is deleted there is still a large component with small diameter in the remaining part. We handle this case by picking a graph such that after the deletion of any set of t vertices we are guaranteed to have a set of almost $n - t$ vertices connected with a small diameter, say k, where we stipulate that $k = O(\log n / \log d)$. In the following we show that well known expander graphs [LPS86] satisfy our requirements. Namely, after deleting an arbitrary set of $t = \lfloor \frac{n-1}{2} \rfloor$ vertices, the remaining set of vertices contains a subgraph of size $\frac{n}{2} - O(\frac{n}{d})$ and of diameter $k = O(\log n / \log d)$, where $d \geq 80$ is a constant. The main result proved in the remainder of this section is therefore as follows:

Theorem 1. *In an LPS expander [LPS86] with $d > 80$, if one deletes half of the vertices then there is a vertex w such that $n/2 - O(n/d)$ of the remaining vertices are at distance $O(\log n / \log d)$ from w.*

We shall use the following result of Alon et al. [AFWZ95].

Theorem 2. *Let $G = (V, E)$ be a d regular graph such that the absolute values of the eigenvalues of its adjacency matrix but the largest are no greater than λ. For a set $B \subseteq V$, $|B| = \mu |V|$, let P be the set of walks of length k (edges) that are all contained in B. Then,*

$$|B| d^k \left(\mu - \frac{\lambda}{d} (1 - \mu) \right)^k \leq |P| \leq |B| d^k \left(\mu + \frac{\lambda}{d} (1 - \mu) \right)^k .$$

Proof (of Theorem 1). Fix a set $B \subseteq V$, $|B| = \frac{1}{2} n$. For a vertex $v \in B$ denote by P_v the set of walks of length k that start at v and never leave B. If follows from the lower bound in Theorem 2 that there is a vertex $w \in B$ for which

$$d^k \left(\frac{1 - \frac{\lambda}{d}}{2} \right)^k \leq |P_w|. \tag{1}$$

Denote by C the set of vertices occurring on walks in P_w. We claim that if

$$k = \frac{\log n}{\log \left(\frac{1 - \frac{\lambda}{d}}{2\left(c + \frac{\lambda}{d}(1-c)\right)} \right)}$$

then $|C| \geq cn$. Otherwise, $|C| < cn$, and from the upper bound in theorem 2 we obtain that

$$|P_w| < cnd^k \left(c + \frac{\lambda}{d}(1-c)\right)^k \tag{2}$$

Combining the lower bound in (1) and the upper bound in (2) we obtain that

$$k < \frac{\log n}{\log\left(\frac{1-\frac{\lambda}{d}}{2\left(c+\frac{\lambda}{d}(1-c)\right)}\right)},$$

in contradiction with our choice of k.

In particular, for $c = 3\left(\frac{\lambda}{d}\right)^2$ we obtain that there is a vertex $w \in B$ such that there are at least $3\left(\frac{\lambda}{d}\right)^2 n$ vertices within distance

$$k = \frac{\log n}{\log\left(\frac{1-\frac{\lambda}{d}}{6\left(\frac{\lambda}{d}\right)^2+2\frac{\lambda}{d}-6\left(\frac{\lambda}{d}\right)^3}\right)} \tag{3}$$

from w in B. If we take LPS expander then $\lambda = 2\sqrt{d-1}$. It is easy to check that for $\lambda = 2\sqrt{d-1}$ and $d > 80$, one has $\frac{1-\frac{\lambda}{d}}{6\left(\frac{\lambda}{d}\right)^2+2\frac{\lambda}{d}} > 1$. Therefore, we obtain that if G is an LPS expander with $d > 80$ then there is a vertex $w \in B$ such that $3\left(\frac{\lambda}{d}\right)^2 n$ of the vertices of B are at distance at most $O(\log n/\log d)$ from w in B. (Notice that the constant hidden by the big-O approaches 2 as d goes to infinity.)

From Lemma 2.4 in Chapter 9 of [ASE92] it follows that if between two sets B and C such that $|B| = bn$ and $|C| = cn$ there is no edge then

$$|C|b^2d^2 \leq \lambda^2 b(1-b)n,$$

so $bc \leq (\frac{\lambda}{d})^2(1-b)$.

From this we get the following consequences:

1. There must be an edge between any set of size $3\left(\frac{\lambda}{d}\right)^2$ and any set of size $\left(\frac{1}{2}-\frac{\lambda}{d}\right)n$ if $\lambda/d < 1/4$.
2. There is an edge between every two sets of size $\frac{\lambda}{d}n$.
3. There is an edge between any set of size $(\frac{1}{2}-\frac{\lambda}{d})n$ and any set of size e/d if $e \geq \left(\frac{\lambda^2}{d}\right)\left(\frac{1/2+\lambda/d}{1/2-\lambda/d}\right)$.

For LPS expanders with $d > 80$, we have that $\lambda/d < 1/4$ and furthermore the condition in 3 holds for $e \geq 11$. Therefore we obtain that the set of vertices within distance $k+3$ from w where k is defined as in 3 is of size at least $(1/2 - 11/d)n$. (Notice that e is smaller and goes to 4 as d goes to infinity.) □

4 *Reconstruct*

The goal of the *Reconstruct* transformation is to take n pieces retrieved from a storage system, up to t of which may be arbitrarily altered from the originally stored values for some document x, and to return the original document x. That is, given a set of pieces $\{r_1, ..., r_n\}$ containing at least $n - t$ original pieces $\{Share(x, i_1), ..., Share(x, i_{n-t})\}$, we want that $Reconstruct(r_1, ..., r_n) = x$ (with high probability). Note that, we need to keep *Reconstruct* feasible as n grows despite the uncertainty concerning up to t of the pieces. Hence, we cannot exhaustively scan all combinations of $n - t$ pieces to find a correct one.

Consider the set $\{r_1, ..., r_n\}$ of retrieved pieces. Every unaltered piece r_i contains the original values $\{V_i^j\}_{j=0,...,k}, H(x)$ which were stored in it. We say that r_i, r_j are *consistent* if they are connected in ST and all levels of verification information in them are consistent. This set induces a subgraph R of ST, with all the edges between inconsistent vertices removed. Our construction of *Share* guarantees the existence of a connected component in R of size $n/2 - O(n/d)$ whose diameter is at most k. Our recovery algorithm finds this set with an expected linear number of steps. Here we let $N^1(I) = N(I)$ be the set of vertices adjacent to vertices in I in the subgraph R of ST. Also we denote by $N^k(I)$ the set of vertices within distance no greater than k to a vertex in I. We prove the following lemma about R:

Lemma 1. *Let I be a set of fully unaltered vertices. Then every vertex in $N^y(I)$, where $y \leq k$, has its first $k - y$ levels of information unaltered.*

Proof. By induction on y. For the basis of the induction, we examine the immediate neighborhood of I. Since the first k-levels of verification information in each $r_i \in I$ are unaltered, for each immediate neighbor $r_j \in N(I)$ the hash values $H(V_0^j), \ldots, H(V_{k-1}^j)$ stored by some $r_i \in I$ are unaltered and hence, (by the cryptographic assumption) $V_0^j, \ldots, V_{k-1}^j$ are unaltered in j.

For the induction step, assume that the lemma holds for $y' < y$. Hence, every vertex in $N^{y-1}(I)$ has its $k-(y-1)$-levels of verification information unaltered. But since $N^y(I) = N(N^{y-1}(I))$ by an argument as for the base case stated above using $I' = N^{y-1}(I)$ and $k' = k - (y - 1)$, we obtain that each vertex in $N(I')$ has $k' - 1 = k - (y - 1) - 1$ levels of verification information unaltered, as desired. □

As an immediate corollary of Lemma 1 we obtain that if K is a connected set in R of diameter no greater than k then either all the IDA shares in K are correct, or all vertices of K are altered.

4.1 The Algorithm

The algorithm *Reconstruct* goes as follows:

1. Let $S = \{r_1, ..., r_n\}$.
2. Let h be the value of $H(x)$ that occurs in $\lceil \frac{n+1}{2} \rceil$ pieces in S.

3. Pick a node $r_i \in S$ at random;
4. If $|N^k(R, r_i)| < n/2 - n/d$ set $S = S \setminus \{r_i\}$ and go back to step 3.
5. Get all the pieces from $N^k(R, r_i)$, reconstruct a document $\hat{x}$ using `IDA` and check that $H(\hat{x}) = h$. If so, return $\hat{x}$ else set $S = S \setminus \{\{r_i\} \cup N^k(R, r_i)\}$ and go back to step 3.

As shown, the storage graph contains, after removing t faulty nodes, a connected component of size $\Theta(n)$ and diameter $O(\log n)$. Hence, in an expected constant number of steps, the retrieval algorithm above will terminate and return the correct response.

5 A Secure Storage System

The application context of our work is a secure storage system. The system consists of a set S of n *servers* denoted $s_1, ..., s_n$, and a distinct set of *clients* accessing them. Correct servers remain alive and follow their specification. Faulty servers however may experience arbitrary (Byzantine) faults, i.e., in the extreme, they can act maliciously in a coordinated way and arbitrarily deviate from their prescribed protocols (ranging from not responding, to changing their behavior and modifying data stored on them). Throughout the run of our protocols, however, we assume a bound $t = \lfloor \frac{n-1}{2} \rfloor$ on the number of faulty servers. Clients are assumed to be correct, and each client may communicate with any server over a reliable, authenticated communication channel. That is, a client c receives a message m from a correct server s if and only if s sent m, and likewise s receives m' from c iff c sent m'. Furthermore, we assume a known upper bound τ on the duration of a round-trip exchange between a client and a correct server, i.e., a client receives a response to message m sent to a correct server s within at most τ delay. In our protocols, we need not make any assumption nor employ communication among the servers.

The storage system provides a pair of protocols, *store* and *retrieve*, whereby clients can store a document x at the servers and retrieve x from them despite arbitrary failures to up to t servers. More precisely, the store and retrieve protocols are as follows:

store: For a client to store x, it sends a message $\langle \mathsf{store}, x \rangle$ to each server in S, and waits for acknowledgment from $n - t$ servers.

retrieve: For a client to retrieve the contents of x, it contacts each server in S with a request $\langle \mathsf{retrieve} \rangle$. It waits for a period of τ to collect a set of responses $A = \{a_s\}_{s \in S}$, where each a_s is either a response of the form $\langle \mathsf{piece}, x_s \rangle$, if s responded in time, or $\perp$ if the timeout expired before s's response was received. The client returns $Reconstruct(A)$ as the retrieved content.

Each server s_i that receives a message $\langle \mathsf{store}, x \rangle$ stores locally the value $Share(x, i)$. And when a server s receives a $\langle \mathsf{retrieve} \rangle$ request it promptly responds with the currently stored piece.

A few points are worthy of noting in this description. First, due to our failure model, a client may receive more than $n - t$ responses to a query, albeit some undetectably corrupted. By assumption, though, the retrieved set A will contain $n - t$ original pieces, and hence, our *Share* and *Reconstruct* algorithms above guarantee that $Reconstruct(A)$ yields the original stored content of x. Second, the store protocol assumes that the computation of each piece is done by each individual server. This is done for simplicity of the exposition. Another possibility would be for a client or some gateway between the clients and servers to first compute the pieces $Share(x, 1), ..., Share(x, n)$ and then send each piece to its corresponding server. The latter form saves computation time by performing it only once at the client (or the gateway), and comes at the expense of increasing the load on the client during a *store* operation. Both forms can be supported (in a similar manner to [GGJ97]), and are orthogonal to the discussion here. Third, during a *retrieve* operation the client may optimize access to the servers, e.g., by contacting an initial set of $n-t$ servers, which will suffice in the normal faultless state of the system, and dynamically increasing it only as needed. Such methods are extensively discussed in the relevant literature on distributed systems and replicated databases, and are not the main focus of the work at hand. Finally, for simplicity, we have assumed that *store* and *retrieve* operations do not overlap, though in practice, concurrency control mechanisms must be applied to enforce this.

6 Discussion

Our research leaves open a number of issues. First, our constants, in particular, the degree d, are rather large, and hence the results are beneficial for very large systems only. We are looking for graph constructions facilitating our methods for smaller system sizes. One such family of candidates are finite projective geometries [A86].

Second, our adversarial assumption is rather strong, namely, fully adaptive malicious adversary, and it might be possible to improve efficiency if we adopt a weaker adversarial model. In particular, one might accept in practice a non-adaptive adversarial model, that is, one that gives the adversary t randomly chosen servers to corrupt. Early on in this work, we envisioned making use of a random graph–$G(n, p)$–in which each edge (i, j) exists with probability p. It is known that for such random graphs, connectivity occurs at $p = (\log n + \omega(n))/n$ (with diameter $d = O(\log n/ \log \log n)$) and that the diameter becomes 2 at $p = \Theta(\sqrt{(\log n)/n})$ (See e.g. [Bollobas85]). Due to the independent selection of edges, any subgraph G' of $G(n, p)$, induced by removal of t randomly selected vertices, is itself a random graph. Hence, it is also connected with a small diameter. This provides a viable solution for smaller system sizes than our current results, albeit for the weaker adversarial model and while incurring an increased probability of error (that is, the probability that the resulting subgraph after removal of faulty vertices is not connected with small diameter). Other candidates to acheive better results in the weaker adversarial model are random regular graphs.

Acknowledgements

We are thankful to the following people for many helpful discussions: Michael Ben-Or, Nati Linial, Eli Shamir and Rebecca Wright.

References

AE90. D. Agrawal and A. El Abbadi. Integrating security with fault-tolerant distributed databases. *Computer Journal* 33(1):71–78, February 1990.

A86. N. Alon. Eigenvalues, geometric expanders, sorting in rounds, and Ramsey theory. *Combinatorica* 6(3):207–219, 1986.

AFWZ95. N. Alon, U. Feige, A. Wigderson and D. Zuckerman. Derandomized graph products. *Computational Complexity* 5:60–75, 1995.

AL96. N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. *IEEE Transactions on Information Theory* 42:1732-1736, 1996.

ASE92. N. Alon, J. Spencer and P. Erdos. The Probabilistic Method. John Wiley & Sons, Inc. 1992.

And96. R. J. Anderson. The Eternity Service. In *Proceedings of Pragocrypt '96*, 1996.

Bollobas85. B. Bollobás. *Random Graphs*, Academic Press, London, 1985.

CL99. M. Castro and B. Liskov. Practical Byzantine fault tolerance. In the *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, USA, February 1999.

GGJ97. J. Garay, R. Gennaro, C. Jutla and T. Rabin. Secure distributed storage and retrieval. In M. Mavronicolas and P. Tsigas, editors, *11th Internation Workshop on Distributed Algorithms, WDAG '97*, pages 275–289, Berlin, 1997. (LNCS 1109).

GRS95. O. Goldreich, R. Rubinfeld, and M. Sudan. Learning polynomials with queries: The highly noisy case. In *Proc. 36th IEEE Symp. on Foundations of Comp. Science*, pages 294–303. IEEE, 1995.

GS99. V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, September 1999.

HT88. M. P. Herlihy and J. D. Tygar. How to make replicated data secure. In *Advances in Cryptology—CRYPTO '87 Proceedings* (Lecture Notes in Computer Science 293), pages 379–391, Springer-Verlag, 1988.

Kra93. H. Krawczyk. Distributed fingerprints and secure information dispersal. In *Proceedings of the 12th ACM Symposium on Principles of Distributed Computing*, pages 207–218, 1993.

Lam79. L. Lamport. How to make a multiprocessor computer that correctly executes multiprocessor programs. *IEEE Transactions on Computers*, C-28(9):690–691, 1979.

LPS86. A. Lubotzky, R. Phillips and P. Sarnak. Explicit expanders and the Ramanujan conjectures. In *Proceedings of the 18th ACM Symposium on the Theory of Computing*, pages 240–246, New York, 1986.

LMS+97. M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman and V. Stemann. Practical loss-resilient codes. In *Proceedings of the 29th Symposium on Theory of Computing*, May 1997.

MR99. D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large scale distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2), 2000.

MK98. D. Mazières and M. F. Kaashoek. Escaping the evils of centralized control with self-certifying pathnames. In the *Proceedings of the 8th ACM SIGOPS European workshop: Support for composing distributed applications*, Sintra, Portugal, September 1998.

Rab89. M. O. Rabin. Efficient dispersal of information for security, load balancing and fault tolerance. *Journal of the ACM*, 36(2):335–348, 1989.

RR00. R. M. Roth and G. Ruckenstern. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transactions on Information Theory*, to appear.

Sch90. F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys* 22(4):299–319, December 1990.

SHA1. FIPS 180–1. Secure Hash Standard. NIST. US Dept. of Commerce, 1995.

Su97. M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

Generating Partial and Multiple Transversals of a Hypergraph*

Endre Boros[1], Vladimir Gurvich[2], Leonid Khachiyan[3], and Kazuhisa Makino[4]

[1] RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway NJ 08854; boros@rutcor.rutgers.edu

[2] Russian Academy of Sciences, Moscow, Russia; gurvich@rutcor.rutgers.edu

[3] Department of Computer Science, Rutgers University, New Brunswick, New Jersey 08903; leonid@cs.rutgers.edu

[4] Department of Systems and Human Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka, 560-8531, Japan; makino@sys.es.osaka-u.ac.jp

Abstract. We consider two natural generalizations of the notion of transversal to a finite hypergraph, arising in data-mining and machine learning, the so called *multiple* and *partial* transversals. We show that the hypergraphs of all multiple and all partial transversals are dual- bounded in the sense that in both cases, the size of the dual hypergraph is bounded by a polynomial in the cardinality and the length of description of the input hypergraph. Our bounds are based on new inequalities of extremal set theory and threshold logic, which may be of independent interest. We also show that the problems of generating all multiple and all partial transversals of an arbitrary hypergraph are polynomial-time reducible to the well-known dualization problem of hypergraphs. As a corollary, we obtain incremental quasi-polynomial-time algorithms for both of the above problems, as well as for the generation of all the minimal Boolean solutions for an arbitrary monotone system of linear inequalities. Thus, it is unlikely that these problems are NP-hard.

1 Introduction

In this paper we consider some problems involving the generation of all subsets of a finite set satisfying certain conditions. The most well-known problem of this type, the generation of all minimal *transversals*, has applications in combinatorics [29], graph theory [18,20,30,32], artificial intelligence [10], game theory [13,14,28], reliability theory [8,28], database theory [1,24,33] and learning theory [2].

Given a finite set V of $n = |V|$ points, and a hypergraph (set family) $\mathcal{A} \subseteq 2^V$, a subset $B \subseteq V$ is called a *transversal* of the family $\mathcal{A}$ if $A \cap B \neq \emptyset$ for all sets

* The research of the first two authors was supported in part by the Office of Naval Research (Grant N00014-92-J-1375), the National Science Foundation (Grant DMS 98-06389), and DIMACS. The research of the third author was supported in part by the National Science Foundation (Grant CCR-9618796). The authors are also thankful to József Beck for helpful discussions.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 588–599, 2000.
© Springer-Verlag Berlin Heidelberg 2000

$A \in \mathcal{A}$; it is called a *minimal transversal* if no proper subset of B is a transversal of $\mathcal{A}$. The hypergraph $\mathcal{A}^d$ consisting of all minimal transversals of $\mathcal{A}$ is called the *dual* (or *transversal*) hypergraph of $\mathcal{A}$. It is easy to see that if $A \in \mathcal{A}$ is not minimal in $\mathcal{A}$, i.e. if $A' \in \mathcal{A}$ for some $A' \subset A$, then $(\mathcal{A} \setminus \{A\})^d = \mathcal{A}^d$. We can assume therefore that all sets in $\mathcal{A}$ are minimal, i.e. that the hypergraph $\mathcal{A}$ is Sperner. (The dual hypergraph $\mathcal{A}^d$ is Sperner by definition.) It is then easy to verify that $(\mathcal{A}^d)^d = \mathcal{A}$ and $\bigcup_{A \in \mathcal{A}} A = \bigcup_{B \in \mathcal{A}^d} B$.

For a subset $X \subseteq V$ let $X^c = V \setminus X$ denote the complement of X, and let $\mathcal{A}^c = \{A^c | A \in \mathcal{A}\}$ be the *complementary* hypergraph of $\mathcal{A}$. Then e.g. $\mathcal{A}^{dc}$ consists of all maximal subsets containing no hyperedge of $\mathcal{A}$, while the hypergraph $\mathcal{A}^{cd}$ consists of all minimal subsets of V which are not contained in any hyperedge of $\mathcal{A}$.

1.1 Multiple Transversals

Given a hypergraph $\mathcal{A} \subseteq 2^V$ and a non-negative weight b_A associated with every hyperedge $A \in \mathcal{A}$, a subset X is called a *multiple transversal* (or *b-transversal*), if $|X \cap A| \geq b_A$ holds for all $A \in \mathcal{A}$. The family of all minimal b-transversals then can also be viewed as the family of support sets of minimal feasible binary solutions to the system of inequalities

$$Ax \geq b, \tag{1}$$

where the rows of $A = A_{\mathcal{A}}$ are exactly the characteristic vectors of the hyperedges $A \in \mathcal{A}$, and the corresponding component of b is equal to b_A. Clearly, $b = (1, 1, ..., 1)$ corresponds to the case of (ordinary) transversals. Viewing the columns of A as characteristic vectors of sets, (1) is also known as a *set covering* problem.

Generalizing further and giving up the binary nature of A as well, we shall consider the family $\mathcal{F} = \mathcal{F}_{A,b}$ of (support sets of) all minimal feasible binary vectors to (1) for a given $m \times n$-matrix A and a given m-vector b. We assume that (1) is a *monotone* system of inequalities: if $x \in \mathbb{B}^n$ satisfies (1) then any vector $y \in \mathbb{B}^n$ such that $y \geq x$ is also feasible, where $\mathbb{B} = \{0, 1\}$. For instance, (1) is monotone if A is non-negative. Note that for a monotone system (1) the dual hypergraph $\mathcal{F}^d = \mathcal{F}^d_{A,b}$ is (the complementary hypergraph of) the collection of (supports of) all maximal infeasible vectors for (1). In the sequel we shall assume that the hypergraph $\mathcal{F}_{A,b}$ is represented by the system (1) and not given explicitly, i.e., by a list of all its hyperedges. In particular, this means that the generation of $\mathcal{F}_{A,b}$ and its dual $\mathcal{F}^d_{A,b}$ are both non-trivial.

1.2 Partial Transversals

Given a hypergraph $\mathcal{A} \subseteq 2^V$ and a non-negative threshold $k < |\mathcal{A}|$, a subset $X \subseteq V$ is said to be a *partial transversal*, or more precisely, a *k-transversal*, to the family $\mathcal{A}$ if it intersects all but at most k of the subsets of $\mathcal{A}$, i.e. if $|\{A \in \mathcal{A} | A \cap X = \emptyset\}| \leq k$. Let us denote by $\mathcal{A}^{d_k}$ the family of all minimal k-transversals

of $\mathcal{A}$. Clearly, 0-transversals are exactly the standard transversals, defined above, i.e. $\mathcal{A}^{d_0} = \mathcal{A}^d$. In what follows we shall assume that the hypergraph $\mathcal{A}^{d_k}$ is represented by a list of all the edges of $\mathcal{A}$ along with the value of k.

Let us define a *k-union* from $\mathcal{A}$ as the union of some k distinct subsets of $\mathcal{A}$, and let $\mathcal{A}^{u_k}$ denote the family of all minimal k-unions of $\mathcal{A}$. In other words, $\mathcal{A}^{u_k}$ is the family of all the minimal subsets of A which contain at least k distinct hyperedges of $\mathcal{A}$. By the above definitions, k-union and k-transversal families both are Sperner (even if the input hypergraph $\mathcal{A}$ is not). It is also easy to see that the families of all minimal k-transversals and $(k+1)$-unions are in fact dual, i.e. , $\mathcal{A}^{d_k} = (\mathcal{A}^{u_{k+1}})^d$ for $k = 0, 1, ..., |\mathcal{A}| - 1$.

2 Dualization

In this paper we shall study problems related to the generation of certain types of transversals of hypergraphs. The simplest and most well-known case is, when a Sperner hypergraph $\mathcal{A}$ is given by an explicit list of its edges, and the problem is the generation of its transversal hypergraph $\mathcal{A}^d$. This problem, known as *dualization*, can be stated as follows:

DUALIZATION$(\mathcal{A}, \mathcal{B})$**:** Given explicitly a finite hypergraph $\mathcal{A}$ and a family of its minimal transversals $\mathcal{B} \subseteq \mathcal{A}^d$, either prove that $\mathcal{B} = \mathcal{A}^d$, or find a new transversal $X \in \mathcal{A}^d \setminus \mathcal{B}$.

Clearly, we can generate all hyperedges of $\mathcal{A}^d$ by initializing $\mathcal{B} = \emptyset$ and iteratively solving the above problem $|\mathcal{A}^d| + 1$ times. Note also that in general, $|\mathcal{A}^d|$ can be exponentially large both in $|\mathcal{A}|$ and $|V|$. For this reason, the complexity of generating $\mathcal{A}^d$ is customarily measured in the input and output sizes. In particular, $\mathcal{A}^d$ is said to be generated in *incremental polynomial time* if DUALIZATION$(\mathcal{A}, \mathcal{B})$ can be solved in time polynomial in $|V|, |\mathcal{A}|$ and $|\mathcal{B}|$.

Dualization is one of those intriguing problems, the true complexity of which is not yet known. For many special classes of hypergraphs it can be solved efficiently. For example, if the sizes of all the hyperedges of $\mathcal{A}$ are limited by a constant r, then dualization can be executed in incremental polynomial time, (see e.g. [5,10]). In the quadratic case, i.e. when $r = 2$, there are even more efficient dualization algorithms that run with *polynomial delay*, i.e. in $poly(|V|, |\mathcal{A}|)$ time, where $\mathcal{B}$ is systematically enlarged from $\mathcal{B} = \emptyset$ during the generation process of $\mathcal{A}^d$ (see e.g. [18,20,32]). Efficient algorithms exist also for the dualization of 2-monotonic, threshold, matroid, read-bounded, acyclic and some other classes of hypergraphs (see e.g. [3,7,9,23,26,27]). Though no incremental polynomial time algorithm is known for the general case, an incremental *quasi-polynomial time* one exists (see [11]). This algorithm solves DUALIZATION$(\mathcal{A}, \mathcal{B})$ in $O(nm) + m^{o(\log m)}$ time, where $n = |V|$ and $m = |\mathcal{A}| + |\mathcal{B}|$ (see also [16] for more detail). We should stress that by the above cited result the dualization problem is very unlikely to be NP-hard.

In this paper we shall mainly consider transversals to hypergraphs not given explicitly, but represented in some implicit way, e.g. generating the partial or

multiple transversals of a hypergraph. To be more precise, we shall say that a Sperner hypergraph $\mathcal{G} \subseteq 2^V$ on $n = |V|$ vertices is represented by a *superset oracle* $\mathfrak{O}$ if for any vertex set $X \subseteq V$, $\mathfrak{O}$ can decide whether or not X contains a hyperedge of $\mathcal{G}$. In what follows, we do not distinguish the superset oracle and the input description $\mathfrak{O}$ of $\mathcal{G}$. We assume that the length $|\mathfrak{O}|$ of the input description of $\mathcal{G}$ is at least n and denote by $T_s = T_s(|\mathfrak{O}|)$ the worst-case running time of the oracle on any superset query "Does X contains a hyperedge of $\mathcal{G}$?". In particular, $\mathfrak{O}$ is polynomial-time if $T_s \leq poly(|\mathfrak{O}|)$. Clearly, $\mathfrak{O}$ also specifies (a superset oracle for) the dual hypergraph $\mathcal{G}^d$.

The main problem we consider then can be formulated as follows:

GEN($\mathcal{G}$): Given a hypergraph $\mathcal{G}$ represented by a polynomial time superset oracle $\mathfrak{O}$, and an explicit list of hyperedges $\mathcal{H} \subseteq \mathcal{G}$, either prove that $\mathcal{H} = \mathcal{G}$, or find a new hyperedge in $\mathcal{G} \setminus \mathcal{H}$.

We list below several simple examples for dual pairs of hypergraphs, represented by a polynomial time superset oracle:
1) Multiple transversals. Let (1) be a monotone system of linear inequalities, and let $\mathcal{G} = \mathcal{F}_{A,b}$ be the hypergraph introduced in Section 1.1. Then the input description $\mathfrak{O}$ is (A, b). Clearly, for any input set $X \subseteq V$, we can decide whether X contains a hyperedge of $\mathcal{F}_{A,b}$ by checking the feasibility of (the characteristic vector of) X for (1).
2) Partial transversals. Let $\mathcal{G} = \mathcal{A}^{d_k}$ be the hypergraph of the minimal k-transversals of a family $\mathcal{A}$, as in Section 1.2. Then $\mathcal{G}$ is given by the threshold value k and a complete list of all hyperedges of $\mathcal{A}$, i.e., $\mathfrak{O} \sim (k, \mathcal{A})$. For a subset $X \subseteq V$, determining whether X contains a hyperedge in $\mathcal{A}^{d_k}$ is equivalent to checking if X is intersecting at least $|\mathcal{A}| - k$ hyperedges of $\mathcal{A}$.
3) Monotone Boolean formulae. Let f be a $(\vee, \wedge)$-formula with n variables and let $\mathcal{G} = \mathcal{A}_f$ be the supporting sets of all the minimal true vectors for f. Then $\mathfrak{O} \sim f$ and the superset oracle checks if (the characteristic vector of) $X \subseteq V$ satisfies f. The dual hypergraph $\mathcal{G}^d$ is the set of all the (complements to the support sets of) maximal false vectors of f.
4) Two terminal connectivity. Consider a digraph $\Gamma = (N, A)$ with a source s and a sink t, and let $\mathcal{G}$ be the set of $s - t$ *paths*, i.e., minimal subsets of arcs that connect s and t. Then $\mathfrak{O} \sim \Gamma$, and for a given arc set $X \subseteq A$, the superset oracle can use breadth-first search to check the reachability of t from s via a path consisting of arcs in X. Note that the dual hypergraph $\mathcal{G}^d$ is the set of all $s - t$ *cuts*, i.e., minimal subsets of arcs that disconnect s and t.
5) Helly's systems of polyhedra. Consider a family of n convex polyhedra $P_i \subseteq \mathbb{R}^r$, $i \in V$, and let $\mathcal{G}$ denote the minimal subfamilies with no point in common. Then $\mathcal{G}^{dc}$ is the family of all maximal subfamilies with a nonempty intersection. (In particular, if $P_1, \ldots, P_n$ are the facets of a convex polytope Q, then these maximal subfamilies are those facets which have a vertex in common, and hence $\mathcal{G}^{dc}$ corresponds to the set of vertices of Q.) We have $\mathfrak{O} \sim (P_1, \ldots, P_n)$ and, given subsets of polytopes $X \subseteq V$, the superset oracle can use linear programming to check whether $\cap_{i \in X} P_i \neq \emptyset$.

3 Main Results

Let us point out first that problems $\textsc{Gen}(\mathcal{G})$ and $\textsc{Gen}(\mathcal{G}^d)$ can be of very different complexity, in general, e.g. it is known that problem $\textsc{Gen}(\mathcal{F}^d_{A,b})$ is NP-hard even for binary matrices A (see [21]). In contrast to this, we can show that the tasks of generating multiple and ordinary transversals are polynomially related.

Theorem 1. *Problem* $\textsc{Gen}(\mathcal{F}_{A,b})$ *is polytime reducible to dualization.*

In particular, for any monotone system of linear inequalities (1), all minimal binary solutions of (1) can be generated in quasi-polynomial incremental time.

Remark 1. Let us add that though generating all maximal infeasible binary points for (1) is NP-hard, there exists a polynomial randomized scheme for nearly uniform sampling from the set of all binary infeasible points for (1). Such a scheme can be obtained by combining the algorithm [19] for approximating the size of set-unions with the rapidly mixing random walk [25] on the binary cube truncated by a single linear inequality. On the other hand, a similar randomized scheme for nearly uniform sampling from within the set of all binary (or all minimal binary) solutions to a given monotone system (1) is unlikely to exist, since it would imply that any NP-complete problem can be solved in polynomial time by a randomized algorithm with arbitrarily small one-sided failure probability. This can be shown, by using the amplification technique of [17], already for systems (1) with two non-zero coefficients per inequality, see e.g. [15] for more detail.

Similarly, we can show that generating (partial) k-transversals is also polynomially equivalent to the generation of ordinary transversals.

Theorem 2. *Problem* $\textsc{Gen}(\mathcal{A}^{d_k})$ *is polytime reducible to dualization.*

Let us add that the dual problem of generating $(k+1)$-unions, $\textsc{Gen}(\mathcal{A}^{u_{k+1}})$ is known to be NP-hard (see [22].)

In the rest of the paper we present first the two main tools for our proofs: In Section 4 we discuss the method of joint generation for a pair of dual hypergraphs defined via a superset oracle, and show that for a polynomial-time superset oracle the above problem reduces to dualization. Next, in Section 5 we present some inequalities of threshold logic and extremal set theory showing that for the above cases the method of joint generation can efficiently be applied, yielding thus Theorems 1 and 2. In Section 6 we present some of the proofs. Due to space limitations, we cannot include all proofs here, and refer the reader to the technical report [6] for further details. Finally, Section 7 discusses some of the related set families arising in data mining and machine learning, and Section 8 contains our concluding remarks.

4 Joint Generation of Dual Bounded Hypergraphs

One of the main ingredients in our proofs is the method of *joint generation* of the edges of a dual pair of hypergraphs:

$\textsc{Gen}(\mathcal{G}, \mathcal{G}^d)$**:** Given hypergraphs $\mathcal{G}$ and $\mathcal{G}^d$, represented by a superset oracle $\mathfrak{O}$, and two explicitly listed set families $\mathcal{A} \subseteq \mathcal{G}$ and $\mathcal{B} \subseteq \mathcal{G}^d$, either prove that $(\mathcal{A}, \mathcal{B}) = (\mathcal{G}, \mathcal{G}^d)$, or find a new set in $(\mathcal{G} \setminus \mathcal{A}) \cup (\mathcal{G}^d \setminus \mathcal{B})$.

For the special case when $\mathcal{A} = \mathcal{G}$ and $\mathfrak{O}$ is the explicit list of all the sets in $\mathcal{G}$, we obtain the dualization problem as stated in Section 2. More generally, as observed in [4,15], problem $\textsc{Gen}(\mathcal{G}, \mathcal{G}^d)$ can be reduced in polynomial time to dualization for any polynomial-time superset oracle $\mathfrak{O}$:

Proposition 1 ([4,15]). *Problem* $\textsc{Gen}(\mathcal{G}, \mathcal{G}^d)$ *can be solved in* $n\,(poly(|\mathcal{A}|, |\mathcal{B}|) + T_s(|\mathfrak{O}|)) + T_{dual}$ *time, where* T_{dual} *denotes the time required to solve problem* $\textsc{Dualization}(\mathcal{A}, \mathcal{B})$.

In particular, for any (quasi-)polynomial-time oracle $\mathfrak{O}$, problem $\textsc{Gen}(\mathcal{G}, \mathcal{G}^d)$ can be solved in quasi-polynomial time. Thus, for each of the 5 examples mentioned above we can jointly generate all the hyperedges of $(\mathcal{G}, \mathcal{G}^d)$ in incremental quasi-polynomial time.

Joint generation may not be efficient however, for solving either of the problems $\textsc{Gen}(\mathcal{G})$ or $\textsc{Gen}(\mathcal{G}^d)$ separately. For instance, as shown in [15], both problems $\textsc{Gen}(\mathcal{G})$ and $\textsc{Gen}(\mathcal{G}^d)$ are NP-hard for examples 3-5. In fact, in example 3 these problems are NP-hard already for $(\vee, \wedge)$-formulae of depth 3. (If the depth is only 2 then the formula is either a CNF or a DNF and we get exactly dualization.) The simple reason is that we do not control which of the families $\mathcal{G} \setminus \mathcal{A}$ and $\mathcal{G}^d \setminus \mathcal{B}$ contain the new hyperedge produced by the algorithm. Suppose, we want to generate $\mathcal{G}$, and the family $\mathcal{G}^d$ is exponentially larger than $\mathcal{G}$. Then, if we are unlucky, we can get hyperedges of $\mathcal{G}$ with exponential delay, while getting large subfamilies of $\mathcal{G}^d$ (which are not needed at all) in between.

Such a problem will not arise and $\textsc{Gen}(\mathcal{G}, \mathcal{G}^d)$ can be used to produce $\mathcal{G}$ efficiently, if the size of $\mathcal{G}^d$ is polynomially limited in the size of $\mathcal{G}$ and in the input size $|\mathfrak{O}|$, i.e. when there exists a polynomial p such that $|\mathcal{G}^d| \leq p(|V|, |\mathfrak{O}|, |\mathcal{G}|)$. We call such Sperner hypergraphs $\mathcal{G}$ *dual-bounded*. For dual-bounded hypergraphs, we can generate both $\mathcal{G}$ and $\mathcal{G}^d$ by calling $\textsc{Gen}(\mathcal{G}, \mathcal{G}^d)$ iteratively $|\mathcal{G}^d| + |\mathcal{G}| \leq poly(|V|, |\mathfrak{O}|, |\mathcal{G}|)$ times, and hence all the hyperedges of $\mathcal{G}$ can be obtained in *total quasi-polynomial* time.

This approach however, may still be inefficient incrementally, i.e., for obtaining a single hyperedge of $\mathcal{G}$ as required in problem $\textsc{Gen}(\mathcal{G})$. It is easy to see that the decision problem: "Given a family $\mathcal{A} \subseteq \mathcal{G}$, determine whether $\mathcal{A} = \mathcal{G}$?" is polynomially reducible to dualization for any dual-bounded hypergraphs represented by a polynomial-time superset oracle. If $\mathcal{A}$ is much smaller than $\mathcal{G}$, getting a new hyperedge in $\mathcal{G} \setminus \mathcal{A}$ still may require exponentially many (in $|\mathcal{A}|$) calls to $\textsc{Gen}(\mathcal{G}, \mathcal{G}^d)$.

Let us call a Sperner hypergraph $\mathcal{G}$ *uniformly dual-bounded* if there exists a polynomial p such that

$$|\mathcal{H}^d \cap \mathcal{G}^d| \leq p(|V|, |\mathfrak{O}|, |\mathcal{H}|) \text{ for all hypergraphs } \mathcal{H} \subseteq \mathcal{G}. \tag{2}$$

Proposition 2. *Problem* $\textsc{Gen}(\mathcal{G})$ *is polytime reducible to dualization for any uniformly dual-bounded hypergraph* $\mathcal{G}$ *defined by a polynomial-time superset oracle.*

Proof. Given a proper subfamily $\mathcal{A}$ of $\mathcal{G}$, we wish to find a new hyperedge $\mathcal{G} \setminus \mathcal{A}$. Start with the given $\mathcal{A}$ and $\mathcal{B} = \emptyset$ and call $\textsc{Gen}(\mathcal{G}, \mathcal{G}^d)$ repeatedly until it outputs a required hyperedge. If this takes t calls, then $t-1$ hyperedges in $\mathcal{G}^d \cap \mathcal{A}^d$ are produced. Since $\mathcal{G}$ is uniformly dual-bounded, we have $t - 1 \leq |\mathcal{A}^d \cap \mathcal{G}^d| \leq p(|V|, |\mathfrak{O}|, |\mathcal{A}|)$, and hence the statement follows by Proposition 1 □

5 Bounding Partial and Multiple Transversals

Theorems 1 and 2 will follow from Proposition 2, by showing that the Sperner hypergraphs $\mathcal{F}_{A,b}$ and $\mathcal{A}^{d_k}$ are both uniformly dual-bounded:

Theorem 3. *For any monotone system (1) of* m *linear inequalities in* n *binary variables, and for any* $\mathcal{H} \subseteq \mathcal{F}_{A,b}$ *we have*

$$|\mathcal{H}^d \cap \mathcal{F}^d_{A,b}| \leq mn|\mathcal{H}| \quad \textit{for any} \quad \mathcal{H} \subseteq \mathcal{F}_{A,b}. \tag{3}$$

In particular, $|\mathcal{F}^d_{A,b}| \leq mn|\mathcal{F}_{A,b}|$ follows. To prove the above theorem we shall use the following lemma.

Lemma 1. *Let* $h : \mathbb{B}^n \to \mathbb{B}$ *be a monotone Boolean function,* $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$*, and* $t \in \mathbb{R}$ *such that the inequality* $wx \stackrel{\text{def}}{=} \sum_{i=1}^n w_i x_i \geq t$ *holds for all binary vectors* $x \in \mathbb{B}^n$ *for which* $h(x) = 1$*. Then, if* $h \not\equiv 0$*, we have*

$$|\max F(h) \cap \{x \mid wx < t\}| \leq \sum_{x \in \min T(h)} ex,$$

where $\max F(h) \subset \mathbb{B}^n$ *is the set of all maximal false points of* h*,* $\min T(h) \subseteq \mathbb{B}^n$ *is the set of all minimal true points of* h*, and* e *is the vector of all ones.*

In particular, $|\max F(h) \cap \{x \mid wx < t\}| \leq n|\min T(h)|$ must hold. If the function h is threshold ($h(x) = 1 \Leftrightarrow wx \geq t$), then $|\max F(h)| \leq n|\min T(h)|$ is a well-known inequality (see [3,9,26,27]). Lemma 1 thus extends this result to arbitrary monotone functions.

For the hypergraph of partial transversals we can prove the following:

Theorem 4. *For any hypergraph* $\mathcal{A} \subseteq 2^V$ *of* $m = |\mathcal{A}|$ *hyperedges, threshold* $0 \leq k \leq m-1$*, and subfamily* $\mathcal{H} \subseteq \mathcal{A}^{d_k}$ *we have*

$$|\mathcal{H}^d \cap \mathcal{A}^{u_{k+1}}| \leq 2|\mathcal{H}|^2 + (m-k-2)|\mathcal{H}|. \tag{4}$$

In particular, $|\mathcal{A}^{u_{k+1}}| \leq 2|\mathcal{A}^{d_k}|^2 + (m-k-2)|\mathcal{A}^{d_k}|$ follows. To prove the above theorem, we need the following combinatorial inequality.

Lemma 2. *Let $\mathcal{A} \subseteq 2^V$ be a hypergraph on $|V| = n$ vertices with $|\mathcal{A}| \geq 2$ hyperedges such that*

$$|A| \geq k+1 \text{ for all } A \in \mathcal{A}, \text{ and } |B| \leq k \text{ for all } B \in \mathcal{A}^{\cap}, \tag{T}$$

where k is a given threshold and $\mathcal{A}^{\cap}$ is the family of all the maximal subsets of V which can be obtained as the intersection of two distinct hyperedges of $\mathcal{A}$. Then

$$|\mathcal{A}| \leq (n-k)|\mathcal{A}^{\cap}|. \tag{5}$$

Note that $\mathcal{A}^{\cap}$ is a Sperner family by its definition, and that condition (T) implies the same for $\mathcal{A}$. Note also that the thresholdness condition (T) is essential for the validity of the lemma – without (T) the size of $\mathcal{A}$ can be exponentially larger than that of $\mathcal{A}^{\cap}$. There are examples (see [6]) for Sperner hypergraphs $\mathcal{A}$ for which $|\mathcal{A}^{\cap}| = n/5$ and $|\mathcal{A}| = 3^{n/5} + 2n/5$ or $|\mathcal{A}^{\cap}| = (n-2)^2/9$ and $|\mathcal{A}| = 3^{(n-2)/3} + 2(n-2)/3$.

The proof of Theorem 4 in the next section makes further use of the following modification of Lemma 2.

Lemma 3. *Let $\mathcal{S} = \{S_1, \ldots, S_\alpha\}$ and $\mathcal{T} = \{T_1, \ldots, T_\beta\}$ be two non-empty multi-hypergraphs on an n-element vertex set V, such that*

$$|S_i| \geq k+1 \text{ for all } i = 1, \ldots, \alpha, \quad \text{and} \quad |T_l| \leq k \text{ for all } l = 1, \ldots, \beta,$$

where $k < n$ is a given threshold. Suppose that for any two distinct indices $1 \leq i < j \leq \alpha$, the intersection $S_i \cap S_j$ is contained in some hyperedge $T_l \in \mathcal{T}$. Then $\mathcal{S}$ is a Sperner hypergraph and

$$\alpha \leq 2\beta^2 + \beta(n-k-2). \tag{6}$$

6 Proofs

Due to space limitations, we include only the proofs of Lemmas 2, 3 and Theorem 4. The interested reader can find all the proofs and some generalizations in the technical report [6].

Proof of Lemma 2. Let $n = |V|$, $m = |\mathcal{A}| \geq 2$ and $p = |\mathcal{A}^{\cap}|$. We wish to show that

$$m \leq (n-k)p.$$

We prove the lemma by induction on k. Clearly, if $k = 0$, then the lemma holds for all n, since in this case the thresholdness condition (T) implies that all sets in $\mathcal{A}$ are pairwise disjoint and consequently, $m \leq n-k$. Assume hence $k \geq 1$.

For a vertex $v \in V$, let $\mathcal{A}_v = \{A \setminus \{v\} \mid A \in \mathcal{A},\ v \in A\}$ be the sets in $\mathcal{A}$ that contain v, restricted to the base set $V \setminus \{v\}$. Next, let $\mathcal{A}_v^{\cap}$ denote the family of all maximal pairwise intersections of distinct subsets in $\mathcal{A}_v$. If $m_v = |\mathcal{A}_v| \geq 2$ then $\mathcal{A}_v$ and $\mathcal{A}_v^{\cap}$ satisfy the assumptions of the lemma with $n' = n-1$ and $k' = k-1$. Hence

$$m_v \leq (n-k)p_v, \tag{7}$$

where $p_v = |\mathcal{A}_v^{\cap}|$. Let $V_1 = \{v \in V \mid m_v = 1\}$ and let $V_2 = \{v \in V | m_v \geq 2\} = \bigcup_{X \in \mathcal{A}^{\cap}} X$ be the union of all the (maximal) intersections in $\mathcal{A}^{\cap}$. Note that in view of (T) we have $s = \max_{X \in \mathcal{A}^{\cap}} |X| \leq k$. Summing up the left-hand sides of inequalities (7) for all $v \in V_2$ we obtain: $\sum_{v \in V_2} m_v = \sum_{A \in \mathcal{A}} |A| - |V_1| \geq (k+1)m - |V_1|$. Summing up the right-hand sides of the same inequalities over $v \in V_2$ we get $(n-k)\sum_{v \in V_2} p_v = (n-k)\sum_{X \in \mathcal{A}^{\cap}} |X| \leq (n-k)ps$. Hence

$$m \leq \frac{|V_1|}{k+1} + \frac{p(n-k)s}{k+1}.$$

Since $|X| = s$ for at least one set in $\mathcal{A}^{\cap}$, we have $|V_1| \leq n - s$. To complete the inductive proof it remains to show that $n - s \leq (k+1-s)(n-k)p$. This follows from the inequality $n - s \leq (k+1-s)(n-k)$. □

Proof of Lemma 3. The fact that $\mathcal{S}$ is Sperner is trivial. Assume w.l.o.g. that $\mathcal{T}$ is also a Sperner hypergraph and that all the hyperedges of $\mathcal{T}$ have cardinality exactly k. Extend V by introducing two new vertices a_l and b_l for every $T_l \in \mathcal{T}$, and let $\mathcal{A}'$ be the hypergraph obtained by adding to $\mathcal{S}$ two new sets $T_l \cup \{a_l\}$ and $T_l \cup \{b_l\}$ for each $l = 1, \ldots, \beta$. The hypergraph $\mathcal{A}'$ has $m' = \alpha + 2\beta$ hyperedges and $n' = n + 2\beta$ vertices. It is easy to see that $(\mathcal{A}')^{\cap} = \mathcal{T}$. Applying Lemma 2 with $p' = |(\mathcal{A}')^{\cap}| = \beta$ and $k' = k$, we obtain $m' \leq (n' - k')p'$, which is equivalent to (6). □

Proof of Theorem 4. Given a hypergraph $\mathcal{A} \subseteq 2^V$ on an n-element vertex set V, let $\phi : 2^V \to 2^{\mathcal{A}}$ be the monotone mapping which assigns to a set $X \subseteq V$ the subset $\phi(X) \subseteq \{1, 2, ..., m\}$ of indices of all those hyperedges of $\mathcal{A}$ which are contained in X, i.e. $\phi(X) = \{i \mid A_i \subseteq X,\ 1 \leq i \leq m\}$. Note that for any two sets $X, Y \subseteq V$ we have the identity

$$\phi(X) \cap \phi(Y) \equiv \phi(X \cap Y). \tag{8}$$

Let $\mathcal{H} = \{H_1, \ldots, H_\beta\} \subseteq 2^V$ be an arbitrary non-empty collection of k-transversals for $\mathcal{A}$. To show that $\mathcal{H}$ satisfies inequality (4), consider the multi-hypergraph $\mathcal{T} = \{\phi(H_1^c), \ldots, \phi(H_\beta^c)\}$. Since each $H_l \in \mathcal{H}$ is a k-transversal to $\mathcal{A}$, the complement of H_l contains at most k hyperedges of $\mathcal{A}$. Hence $|\phi(H_l^c)| \leq k$, i.e., the size of each hyperedge of $\mathcal{T}$ is at most k.

Let the hypergraph $\mathcal{H}^d \cap \mathcal{A}^{u_{k+1}}$ consist of α hyperedges, say $\mathcal{H}^d \cap \mathcal{A}^{u_{k+1}} = \{X_1, \ldots, X_\alpha\} \subseteq 2^V$. Consider the multi-hypergraph $\mathcal{S} = \{\phi(X_1), \ldots, \phi(X_\alpha)\}$. Note that $X_1, \ldots, X_\alpha$ are $(k+1)$-unions, and hence each hyperedge of $\mathcal{S}$ has size at least $k+1$.

Let us now show that for any two distinct indices $1 \leq i < j \leq \alpha$, the intersection of the hyperedges $\phi(X_i)$ and $\phi(X_j)$ is contained in a hyperedge of $\mathcal{T}$. In view of (8) we have to show that $\phi(X_i \cap X_j)$ is contained in some hyperedge of $\mathcal{T}$. To this end, observe that $\mathcal{H}^d \cap \mathcal{A}^{u_{k+1}}$ is a Sperner hypergraph and hence $X_i \cap X_j$ is a proper subset of X_i. However, $X_i \in \mathcal{H}^d \cap \mathcal{A}^{u_{k+1}}$ is a minimal transversal to $\mathcal{H}$. For this reason, $X_i \cap X_j$ misses a hyperedge of $\mathcal{H}$, say H_l. Equivalently, $X_i \cap X_j \subseteq H_l^c$ which implies $\phi(X_i \cap X_j) \subseteq \phi(H_l^c) \in \mathcal{T}$. Now inequality (4) and Theorem 4 readily follow from Lemma 3. □

7 Related Set-Families

The notion of frequent sets appears in the data-mining literature, see [1,24], and can be related naturally to the families considered above. More precisely, following a definition of [31], given a $(0,1)$-matrix and a threshold k, a subset of the columns is called *frequent* if there are at least k rows having a 1 entry in each of the corresponding positions. The problems of generating all *maximal frequent* sets and their duals, the so called *minimal infrequent* sets (for a given binary matrix) were proposed, and the complexity of the corresponding decision problems were asked in [31]. Results of [22] imply that it is NP-hard to determine whether a family of maximal frequent sets is incomplete, while our results prove that generating all minimal infrequent sets polynomially reduces to dualization.

Since the family $\mathcal{A}^{d_k}$ consists of all the minimal k-transversals to $\mathcal{A}$, i.e. subsets of V which are disjoint from at most k hyperedges of $\mathcal{A}$, the family $\mathcal{A}^{cd_k}$ consists of all the minimal subsets of V which are contained in at most k hyperedges of $\mathcal{A}$. It is easy to recognize that these are the *minimal infrequent sets* in a matrix, the rows of which are the characteristic vectors of the hyperedges of $\mathcal{A}$. Furthermore, the family $\mathcal{A}^{d_k c}$ consists of all the maximal subsets of V, which are supersets of at most k hyperedges of $\mathcal{A}$.

Due to our results above, all these families can be generated e.g. in incremental quasi-polynomial time.

In the special case, if $\mathcal{A}$ is a quadratic set-family, i.e. if all hyperedges of $\mathcal{A}$ are of size 2, the family $\mathcal{A}$ can also be interpreted as the edge set of a graph G on vertex set V. Then, $\mathcal{A}^{d_k c}$ is also known as the family of the so called *fairly independent sets* of the graph G, i.e. all the vertex subsets which induce at most k edges (see [31].)

As it was defined above, the family $\mathcal{A}^{u_k}$ consists of all the minimal k-unions of $\mathcal{A}$, i.e. all minimal subsets of V which contain at least k hyperedges of $\mathcal{A}$, and hence the family $\mathcal{A}^{cu_k}$ consists of all the minimal subsets which contain at least k hyperedges of $\mathcal{A}^c$. Thus, the family $\mathcal{A}^{cu_k c}$ consists of all the maximal k-intersections, i.e. maximal subsets of V which are subsets of at least k hyperedges of $\mathcal{A}$. These sets can be recognized as the *maximal frequent sets* in a matrix, the rows of which are the characteristic vectors of the hyperedges of $\mathcal{A}$. Finally, the family $\mathcal{A}^{u_k c}$ consists of all the maximal subsets of V which are disjoint from at least k hyperedges of $\mathcal{A}$.

As it follows from the mentioned results (see e.g. [21,22]), generating all hyperedges for each of these families is NP-hard , unless k (or $|\mathcal{A}|-k$) is bounded by a constant.

8 General Closing Remarks

In this paper we considered the problems of generating all partial and all multiple transversals. Both problems are formally more general than dualization, but in fact both are polynomially equivalent to it because the corresponding pairs of hypergraphs are uniformly dual-bounded.

It might be tempting to look for a common generalization of these notions, and of these results. However, the following attempt to combine partial and multiple transversals fails. For instance, generating all the minimal partial binary solutions to a system of inequalities $Ax \geq b$ is NP-hard, even if A is binary and $b = (2, 2, ..., 2)$. To show this we can use arguments analogous to those of [21,22]. Consider the well-known NP-hard problem of determining whether a given graph $G = (V, E)$ contains an independent vertex set of size t, where $t \geq 2$ is a given threshold. Introduce $|V| + 1$ binary variables x_0 and x_v, $v \in V$, and write t inequalities $x_u + x_v \geq 2$ for each edge $e = (u, v) \in E$, followed by one inequality $x_0 + x_v \geq 2$, for each $v \in V$. It is easily seen that the characteristic vector of any edge $e = (u, v)$ is a minimal binary solution satisfying at least t inequalities of the resulting system. Deciding whether there are other minimal binary solutions satisfying $\geq t$ inequalities of the system is equivalent to determining whether G has an independent set of size t.

References

1. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen and A. I. Verkamo, Fast discovery of association rules, In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy eds., *Advances in Knowledge Discovery and Data Mining*, 307-328, AAAI Press, Menlo Park, California, 1996.
2. M. Anthony and N. Biggs, *Computational Learning Theory*, Cambridge University Press, 1992.
3. P. Bertolazzi and A. Sassano, An $O(mn)$ time algorithm for regular set-covering problems, *Theoretical Computer Science* 54 (1987) 237-247.
4. J. C. Bioch and T. Ibaraki, Complexity of identification and dualization of positive Boolean functions, *Information and Computation* 123 (1995) 50-63.
5. E. Boros, V. Gurvich, and P.L. Hammer, Dual subimplicants of positive Boolean functions, *Optimization Methods and Software*, 10 (1998) 147-156.
6. E. Boros, V. Gurvich, L. Khachiyan and K.Makino, Dual-Bounded Hypergraphs: Generating Partial and Multiple Transversals, DIMACS Technical Report 99-62, Rutgers University, 1999. (http://dimacs.rutgers.edu/TechnicalReports/1999.html)
7. E. Boros, P. L. Hammer, T. Ibaraki and K. Kawakami, Polynomial time recognition of 2-monotonic positive Boolean functions given by an oracle, *SIAM Journal on Computing* 26 (1997) 93-109.
8. C. J. Colbourn, *The combinatorics of network reliability*, Oxford University Press, 1987.
9. Y. Crama, Dualization of regular Boolean functions, *Discrete Applied Mathematics*, 16 (1987) 79-85.
10. T. Eiter and G. Gottlob, Identifying the minimal transversals of a hypergraph and related problems, *SIAM Journal on Computing*, 24 (1995) 1278-1304.
11. M. L. Fredman and L. Khachiyan, On the complexity of dualization of monotone disjunctive normal forms. *Journal of Algorithms*, 21 (1996) 618-628.
12. M. R. Garey and D. S. Johnson, *Computers and Intractability*, Freeman, New York, 1979.
13. V. Gurvich, To theory of multistep games, *USSR Comput. Math. and Math Phys.* **13-6** (1973) 1485-1500.

14. V. Gurvich, Nash-solvability of games in pure strategies, *USSR Comput. Math and Math. Phys.* **15-2** (1975) 357-371.
15. V. Gurvich and L. Khachiyan, On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions, *Discrete Applied Mathematics*, 97 (1999) 363-373.
16. V. Gurvich and L. Khachiyan, On the frequency of the most frequently occurring variable in dual DNFs, *Discrete Mathematics* 169 (1997) 245-248.
17. M. R. Jerrum, L. G. Valiant and V. V. Vazirani, Random generation of combinatorial structures from a uniform distribution *Theoretical Computer Science* 43 (1986) 169-188.
18. D. S. Johnson, M. Yannakakis and C. H. Papadimitriou, On generating all maximal independent sets, *Information Processing Letters*, 27 (1988) 119-123.
19. R. Karp and M, Luby, Monte-Carlo algorithms for enumeration and reliability problems, in *Proc. 24th IEEE Symp. on Foundations of Computer Science* (1983) 56-64.
20. E. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan, Generating all maximal independent sets: NP-hardness and polynomial-time algorithms, *SIAM Journal on Computing*, 9 (1980) 558-565.
21. K. Makino and T. Ibaraki, Interor and exterior functions of Boolean functions, *Discrete Applied Mathematics*, 69 (1996) 209-231.
22. K. Makino and T. Ibaraki, Inner-core and outer-core functions of partially defined Boolean functions, *Discrete Applied Mathematics*, 96-97 (1999), 307-326.
23. K. Makino and T. Ibaraki, A fast and simple algorithm for identifying 2-monotonic positive Boolean functions, *Journal of Algorithms*, 26 (1998) 291-305.
24. H. Mannila and K. J. Räihä, Design by example: An application of Armstrong relations, *Journal of Computer and System Science* 22 (1986) 126-141.
25. B. Morris and A. Sinclair, Random walks on truncated cubes and sampling 0-1 knapsack problem, in *Proc. 40th IEEE Symp. on Foundations of Computer Science* (1999) 230-240.
26. U. N. Peled and B. Simeone, Polynomial-time algorithm for regular set-covering and threshold synthesis, *Discrete Applied Mathematics* 12 (1985) 57-69.
27. U. N. Peled and B. Simeone, An $O(nm)$-time algorithm for computing the dual of a regular Boolean function, *Discrete Applied Mathematics* 49 (1994) 309-323.
28. K. G. Ramamurthy, *Coherent Structures and Simple Games*, Kluwer Academic Publishers, 1990.
29. R. C. Read, Every one a winner, or how to avoid isomorphism when cataloging combinatorial configurations, *Annals of Discrete Mathematics* 2 (1978) 107-120.
30. R. C. Read and R. E. Tarjan, Bounds on backtrack algorithms for listing cycles, paths, and spanning trees, *Networks* 5 (1975) 237-252.
31. R. H. Sloan, K. Takata, G. Turan, On frequent sets of Boolean matrices, *Annals of Mathematics and Artificial Intelligence* 24 (1998) 1-4.
32. S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, A new algorithm for generating all maximal independent sets, *SIAM Journal on Computing*, 6 (1977) 505-517.
33. J. D. Ullman, *Principles of Database and Knowledge Base Systems*, Vols. 1 and 2, Computer Science Press, 1988.

Revisiting the Correspondence between Cut Elimination and Normalisation

José Espírito Santo*

Laboratory for Foundations of Computer Science
Division of Informatics, The University of Edinburgh, James Clerk Maxwell Building,
The King's Buildings, Mayfield Road, Edinburgh EH9 3JZ, Scotland UK
jes@dcs.ed.ac.uk

Abstract. Cut-free proofs in Herbelin's sequent calculus are in 1-1 correspondence with normal natural deduction proofs. For this reason Herbelin's sequent calculus has been considered a privileged middle-point between L-systems and natural deduction. However, this bijection does not extend to proofs containing cuts and Herbelin observed that his cut-elimination procedure is not isomorphic to β-reduction.
In this paper we equip Herbelin's system with rewrite rules which, at the same time: (1) complete in a sense the cut elimination procedure firstly proposed by Herbelin; and (2) perform the intuitionistic "fragment" of the tq-protocol - a cut-elimination procedure for classical logic defined by Danos, Joinet and Schellinx. Moreover we identify the subcalculus of our system which is isomorphic to natural deduction, the isomorphism being with respect not only to proofs but also to normalisation.
Our results show, for the implicational fragment of intuitionistic logic, how to embed natural deduction in the much wider world of sequent calculus and what a particular cut-elimination procedure normalisation is.

1 Introduction

In his paper about a "λ-calculus structure" isomorphic to a "Gentzen-style sequent calculus structure"[6], Herbelin proposed to define a λ-like calculus corresponding to a LJ-like sequent calculus in the same way as λ-calculus corresponds to natural deduction.

Herbelin starts by refining the simplest, many-one assignment of terms to sequent calculus proofs, usually denoted by φ and that comes from the theory of the relationship between sequent calculus and natural deduction [10,15,9,11,3]. The refinement is to consider a restriction of LJ called LJT (respec. a term calculus called $\overline{\lambda}$-calculus) whose cut-free proofs (respec. cut-free terms) are in 1-1 correspondence with normal natural deduction proofs (respec. n ormal λ-terms).

* The author is supported by Fundação para a Ciência e Tecnologia, Portugal.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 600–611, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Dyckhoff and Pinto [2,3] showed the merits of the *cut-free* fragment of LJT as a proof-theoretical tool and emphasized its privileged intermediate position between sequent calculus and natural deduction. The purpose of this paper is to define an intermediate system of this kind for proofs possibly containing cuts and a cut-elimination procedure which together give an isomorphic copy of natural deduction in sequent calculus format, with respect not only to proofs *but also to proof normalisation.*

Full LJT is not the solution to this problem. The bijection with natural deduction does not extend to proofs with cuts and Herbelin observed that his cut-elimination procedure fails to implement full β-reduction (it just implements a strategy).

The $\overline{\lambda}$-calculus includes an operator of explicit substitution so that local steps of cut permutation can be written as elementary steps of substitution propagation (calculi of explicit substitution for similar purposes can be found in [14,5,13]). Instead of making substitution explicit, we perform the complete upwards permutation of a cut in a single step of reduction by a global operation. This is inspired in the so-called tq-protocol, a cut-elimination procedure for classi cal "coloured" proofs defined by Danos, Joinet and Schellinx [1].

We equip LJT with a reduction procedure of this kind which completes, in a sense, LJT's original procedure, obtaining a sequent calculus and corresponding λ-calculus which we call HJ^+ and λ_H^+, respectively. We prove that HJ^+ is just performing the intuitionistic "fragment" of the tq-protocol and that the typable subcalculus of λ_H^+ is strongly normalising and confluent. Furthermore, we identify natural subsystems HJ and λ_H such that HJ (respe c. λ_H) is isomorphic, in the strong sense required above, to NJ (respec. λ). In particular, both λ_H^+ and λ_H implement full β-reduction.

The reader finds in Table 1 (where *inc* stands for inclusion) a map of systems and translations which will appear in the following.

Notations and Terminology

We just treat intuitionistic implicational logic (implication written as $\supset$). Barendregt's convention applies to all calculi in this paper. A *context* is a *consistent* set of *declarations* $x : A$. By consistent we mean that if $x : A$ and $x : B$ are in a context, then $A = B$. Contexts are ranged over by Γ. We write $x \in \Gamma$ meaning $x : A \in \Gamma$ for some A. $\Gamma, x : A$ denotes the *consistent union* $\Gamma \cup \{x : A\}$, which means that, if x is already declared in Γ, then it is declared with type A.

We call *left* (respec. *right*) subderivation of a cut instance the subderivation in which the cutformula occurs in the RHS (respec. LHS) of its endsequent. Such cutformula is called the *left* (respec. *right*) cutformula of that instance.

2 Background

2.1 Herbelin's LJT and $\overline{\lambda}$-Calculus

We refer to [6] for details about LJT and $\overline{\lambda}$. We adopt some simplification of syntax introduced in [2] (but the numbering of cuts is different!). Table 2 presents

Table 1. Map of systems and translations

$$
\begin{tikzcd}
 & LJT/\overline{\lambda} \arrow[d, "inc"] & & \\
 & LJT^{+}/\overline{\lambda}^{+} \arrow[d, dotted] & & \\
LJ^{t} \arrow[r, shift right, "\overline{\overline{\varphi}}"'] & HJ^{+}/\lambda_{H}^{+} \arrow[l, shift right, "\overline{\overline{\rho}}"'] \arrow[r, shift right, "(_)^{-}"'] & HJ/\lambda_{H} \arrow[l, shift right, "inc"'] \arrow[r, shift right, "\Theta"'] & NJ/\lambda \arrow[l, shift right, "\Psi"']
\end{tikzcd}
$$

the syntax and typing rules of $\overline{\lambda}$ (and thus the inference rules of LJT) and the reduction rules of $\overline{\lambda}$ which define the cut-elimination procedure of LJT (*Der* stands for Dereliction).

In $\overline{\lambda}$ there are two kinds of expressions: terms and lists of terms. The term $x[t_1, ..., t_n]$ can be seen as the λ-term $(...(xt_1)...t_n)$ but with the advantage of having the head-variable at the surface. $t\{x := v\}$ and $l\{x := v\}$ are explicit substitution operators and ll' is an explicit append of list (cf. the reduction rules). Notice that in $t\{x := v\}$ and $l\{x := v\}$, x is bound and $x \notin FV(v)$.

There are two kinds of derivable sequents: $\Gamma; - \vdash t : B$ and $\Gamma; A \vdash l : B$. In both there is a distinguished position in the LHS called *stoup*. The crucial restriction of LJT is that the rule $L \supset$ introduces $A \supset B$ in the stoup and B has to be in the stoup of the right subderivation's endsequent. Forget for a second rules cut_2 and cut_4. In this case (in particular in cut-free LJT), besides Ax, no rule can introduce a formula in the stoup and thus the last rule o f the right subderivation of an instance of $L \supset$ is again $L \supset$ and so on until Ax is reached.

There are two kinds of cuts (head-cut and mid-cut) according to whether the right cutformula is in the stoup or not. Notice that in the reduction rules there are no permutation of cuts.

2.2 LJ^t and the Intuitionistic "tq-Protocol"

Table 3 presents the sequent calculus LJ^t and a corresponding, nameless term calculus in which a cut-elimination procedure is expressed.

We leave to the reader to provide the definitions of free and bound variable in a term L. The idea is that, in $\mathsf{L}(x, L_1, (y)L_2)$, x occurs free and y bound. By Barendregt's convention, neither y occurs free in L_1 nor x occurs bound in L_1 or

Table 2. Herbelin's LJT and $\overline{\lambda}$-calculus

$$u, v, t ::= xl \mid \lambda x.t \mid tl \mid t\{x := v\}$$
$$l, l' ::= [] \mid t :: l \mid ll' \mid l\{x := v\}$$

$$Ax\frac{}{\Gamma; A \vdash [] : A} \qquad Der\frac{\Gamma; A \vdash l : B}{\Gamma, x : A; - \vdash xl : B}$$

$$L\supset\ \frac{\Gamma; - \vdash t : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash t :: l : C} \qquad R\supset\ \frac{\Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash \lambda x.t : A \supset B} x \notin \Gamma$$

mid-cuts

$$cut_1\frac{\Gamma; - \vdash v : A \ \Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash t\{x := v\} : B} x \notin \Gamma \qquad cut_2\frac{\Gamma; - \vdash v : A \ \Gamma, x : A; C \vdash l : B}{\Gamma; C \vdash l\{x := v\} : B} x \notin \Gamma$$

head-cuts

$$cut_3\frac{\Gamma; - \vdash t : A \quad \Gamma; A \vdash l : B}{\Gamma; - \vdash tl : B} \qquad cut_4\frac{\Gamma; C \vdash l : A \quad \Gamma; A \vdash l' : B}{\Gamma; C \vdash ll' : B}$$

$$\begin{array}{lrl}
(\overline{\lambda}11) & (\lambda x.t)(u :: l) & \to t\{x := u\}l \\
(\overline{\lambda}12) & t[] & \to t \\
 & & \\
(\overline{\lambda}21) & (xl)l' & \to x(ll'),\ l' \neq [] \\
 & & \\
(\overline{\lambda}31) & (u :: l)l' & \to u :: (ll') \\
(\overline{\lambda}32) & []l & \to l \\
 & & \\
(\overline{\lambda}41) & (xl)\{x := v\} & \to vl\{x := v\} \\
(\overline{\lambda}42) & (yl)\{x := v\} & \to yl\{x := v\}, y \neq x \\
(\overline{\lambda}43) & (\lambda y.u)\{x := v\} & \to \lambda y.u\{x := v\} \\
 & & \\
(\overline{\lambda}51) & (u :: l)\{x := v\} & \to u\{x := v\} :: l\{x := v\} \\
(\overline{\lambda}52) & []\{x := v\} & \to []
\end{array}$$

L_2, although x may occur free in L_1 or L_2 (meaning that an implicit contraction is happening).

The cut-elimination procedure is a "fragment" of the so-called tq-protocol, a strongly normalising and confluent procedure for classical, "coloured" proofs defined in [1]. To be precise, it is the restriction of the tq-protocol to intuitionistic, t-coloured proofs in which an "orientation" of the multiplicative connective $\supset$ has been fixed.

Table 3. LJ^t

$L ::= \mathsf{Ax}(x) \mid \mathsf{Cut}(L,(y)L) \mid \mathsf{L}(x,L,(y)L) \mid \mathsf{R}((x)L)$
$Ax\,\dfrac{}{\Gamma, x : A \vdash \mathsf{Ax}(x) : A}$ $L\supset\ \dfrac{\Gamma \vdash L_1 : A \quad \Gamma, y : B \vdash L_2 : C}{\Gamma, x : A \supset B \vdash \mathsf{L}(x, L_1, (y)L_2) : C}, y \notin \Gamma$ $R\supset\ \dfrac{\Gamma, x : A \vdash L : B}{\Gamma \vdash \mathsf{R}((x)L) : A \supset B} x \notin \Gamma$ $Cut\,\dfrac{\Gamma \vdash L_1 : A \quad \Gamma, y : A \vdash L_2 : C}{\Gamma \vdash \mathsf{Cut}(L_1, (y)L_2) : C} y \notin \Gamma$
Structural step S_1: $\mathsf{Cut}(L_1,(x)L_2) \to L_2[L_1/x]$, if x is not freshly and logically introduced in L_2 Structural step S_2: $\mathsf{Cut}(L_1,(x)L_2) \to L_1[(x)L_2/-]$, if x is freshly and logically introduced in L_2 and $L_1 \neq \mathsf{R}((z)L_0)$ all z, L_0 Logical step: $\mathsf{Cut}(\mathsf{R}((z)L_0),(x)\mathsf{L}(x,L_1,(y)L_2))$ $\to \mathsf{Cut}(\mathsf{Cut}(L_1,(z)L_0),(y)L_2)$, if x is freshly introduced in $\mathsf{L}(x,L_1,(y)L_2))$ where $\mathsf{Ax}(x)[L/x] = L$ $\mathsf{Ax}(y)[L/x] = \mathsf{Ax}(y), y \neq x$ $\mathsf{L}(x,L',(z)L'')[L/x] = \mathsf{Cut}(L,(x)\mathsf{L}(x,L'[L/x],(z)L''[L/x])$ $\mathsf{L}(y,L',(z)L'')[L/x] = \mathsf{L}(y,L'[L/x],(z)L''[L/x]), y \neq x$ $\mathsf{R}((y)L')[L/x] = \mathsf{R}((y)L'[L/x])$ $\mathsf{Cut}(L',(y)L'')[L/x] = \mathsf{Cut}(L'[L/x],(y)L''[L/x])$ $\mathsf{Ax}(y)[(x)L/-] = L[y/x]$ $\mathsf{L}(y,L',(z)L'')[(x)L/-] = \mathsf{L}(y,L',(z)L''[(x)L/-])$ $\mathsf{R}((y)L')[(x)L/-] = \mathsf{Cut}(\mathsf{R}((y)L'),(x)L)$ $\mathsf{Cut}(L',(y)L'')[(x)L/-] = \mathsf{Cut}(L',(y)L''[(x)L/-])$

Roughly, the protocol works as follows: a cut is firstly permuted upwards through its right subderivation (structural step S_1) and then through its left subderivation (structural step S_2) until it becomes a *logical* cut, to which the logical step applies, giving rise to new cuts of lower degree. A logical cut is a cut whose both cutformulas are *freshly and logically introduced*, *i.e.* introduced by a logical rule ($L\supset$ or $R\supset$) without implicit contraction. An equivalent description of step S_1 (respec. step S_2) is: to push the left (respec. right) subderivation upwards through the "tree of ancestors" [1] of the right (respec. left) cutformula.

The operations $L_2[L_1/x]$ and $L_1[(x)L_2/-]$ implement the structural steps S_1 and S_2, respectively, and are inspired in the operations of substitution and co-substitution defined by Urban and Bierman in [12].

3 HJ^+ and the λ_H^+-Calculus

We refer to Table 4 for the definition of HJ^+ and λ_H^+. The motivation for these systems rests in the following observations.

The "life cycle" of a cut in LJT has three stages. It starts as a mid-cut and the first stage is a upwards permutation through its right subderivation, performed by rules $\overline{\lambda}4i$ and $\overline{\lambda}5j$. The goal is to generate head-cuts (see rule $\overline{\lambda}41$). The operation *subst* performs this permutation in a single step. In doing so, cuts of the form $l\{x := v\}$ become "internal" to this process and hence are not needed in the syntax. Now observe that in LJT such pe rmutation of a mid-cut can complete only if, in its course, we do not need to permute this mid-cut with another cut. This is why, in the definition of *subst*, extra clauses occur corresponding to the permutations

$$\begin{array}{ll} (\overline{\lambda}44) & (tl)\{x := v\} \to t\{x := v\}l\{x := v\} \ , \\ (\overline{\lambda}45) & t\{y := u\}\{x := v\} \to t\{x := v\}\{y := u\{x := v\}\} \ . \end{array}$$

Let us return to the head-cuts generated by the first stage. Notice that in a head-cut vl, if $l \neq []$ then its right cutformula is freshly and logically introduced. Such a cut is permuted upwards through its left subderivation by the rules $\overline{\lambda}21$ and $\overline{\lambda}3i$, generating along the way $\overline{\lambda}11$-redexes, *i.e.* logical cuts in the LJ^t sense. The last stage of the "life cycle" of these logical cuts is $\overline{\lambda}11$-reduction, by which cuts of l ower degree are generated.

Again the operation *insert* performs in a single step the permutations of the second stage and cuts ll' become "internal" and thus superfluous in the syntax. Extra clauses in *insert*'s definition correspond to the permutations

$$\begin{array}{ll} (\overline{\lambda}22) & (tl')l \to t(l'l) \ , \\ (\overline{\lambda}23) & (t\{x := v\})l \to (tl)\{x := v\} \ . \end{array}$$

Define LJT^+ as LJT plus the four new reduction rules just presented. We leave to the reader the formalisation of the obvious relations between LJT, LJT^+ and HJ^+.

On the other hand, it should be clear that reductions $\to_{m(id)}$ and $\to_{h(ead)}$ in HJ^+ have a strong connection with the structural steps S_1 and S_2, respectively,

Table 4. HJ^+ and λ_H^+-calculus

$$u, v, t ::= xl \mid \lambda x.t \mid tl \mid t\{x := v\}$$
$$l, l' ::= [] \mid t :: l$$

$$Ax\frac{}{\Gamma; A \vdash [] : A} \qquad Der\frac{\Gamma; A \vdash l : B}{\Gamma, x : A; - \vdash xl : B}$$

$$L\supset \ \frac{\Gamma; - \vdash t : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash t :: l : C} \qquad R\supset \ \frac{\Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash \lambda x.t : A \supset B} x \notin \Gamma$$

$$mid - cut\,\frac{\Gamma; - \vdash v : A \quad \Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash t\{x := v\} : B} x \notin \Gamma$$

$$head - cut\,\frac{\Gamma; - \vdash t : A \quad \Gamma; A \vdash l : B}{\Gamma; - \vdash tl : B}$$

$$\begin{array}{rl}
(mid) & t\{x := v\} \to subst(v, x, t) \\
(head) & tl \to insert(l, t), \text{ if } t \text{ is not a } \lambda\text{-abstraction or } l = [] \\
(log) & (\lambda x.t)(u :: l) \to t\{x := u\}l
\end{array}$$

where

$$\begin{aligned}
subst(v, x, x[]) &= v \\
subst(v, x, xl) &= v\, subst(v, x, l), l \neq [] \\
subst(v, x, yl) &= y\, subst(v, x, l), y \neq x \\
subst(v, x, \lambda y.t) &= \lambda y.subst(v, x, t) \\
subst(v, x, tl) &= subst(v, x, t) subst(v, x, l) \\
subst(v, x, t\{y := u\}) &= subst(v, x, t)\{y := subst(v, x, u)\}
\end{aligned}$$

$$\begin{aligned}
subst(v, x, u :: l) &= subst(v, x, u) :: subst(v, x, l) \\
subst(v, x, []) &= []
\end{aligned}$$

$$\begin{aligned}
insert([], t) &= t \\
insert(l, xl') &= x\, append(l', l),\ l \neq [] \\
insert(l, \lambda x.t) &= (\lambda x.t)l,\ l \neq [] \\
insert(l, tl') &= t\, append(l', l),\ l \neq [] \\
insert(l, t\{x := v\}) &= insert(l, t)\{x := v\},\ l \neq []
\end{aligned}$$

$$\begin{aligned}
append(t :: l, l') &= t :: append(l, l') \\
append([], l') &= l'
\end{aligned}$$

of LJ^t and that, roughly (but not exactly), a mid-cut is a S_1-redex and a head-cut is a S_2-redex. This is formalised by defining a map $\overline{\overline{\rho}} : HJ^+ \to LJ^t$ as in Table 5 (where $z \notin FV(l)$ in the second and last clauses of the definition of $\overline{\overline{\rho}}$).

Table 5. Translations $\overline{\overline{\rho}}$ and $\overline{\overline{\varphi}}$

$\overline{\overline{\rho}}(x[]) = \mathsf{Ax}(x)$ $\overline{\overline{\rho}}(x(t :: l)) = \mathsf{L}(x, \overline{\overline{\rho}}(t), (z)\overline{\overline{\rho}}(zl))$ $\overline{\overline{\rho}}(\lambda x.t) = \mathsf{R}((x)\overline{\overline{\rho}}(t))$ $\overline{\overline{\rho}}(t\{x := v\}) = \mathsf{Cut}(\overline{\overline{\rho}}(v), (x)\overline{\overline{\rho}}(t))$ $\overline{\overline{\rho}}(tl) = \mathsf{Cut}(\overline{\overline{\rho}}(t), (z)\overline{\overline{\rho}}(zl))$	$\overline{\overline{\varphi}}(\mathsf{Ax}(x)) = x[]$ $\overline{\overline{\varphi}}(\mathsf{R}((x)L)) = \lambda x.\overline{\overline{\varphi}}(L)$ $\overline{\overline{\varphi}}(\mathsf{L}(x, L_1, (y)L_2)) = \overline{\overline{\varphi}}(L_2)\{y := x[\overline{\overline{\varphi}}(L_1)]\}$ $\overline{\overline{\varphi}}(\mathsf{Cut}(L_1, (y)L_2)) = \overline{\overline{\varphi}}(L_2)\{y := \overline{\overline{\varphi}}(L_1)\}$

Lemma 1. $\overline{\overline{\rho}}(subst(v, x, t)) = \overline{\overline{\rho}}(t)[\overline{\overline{\rho}}(v)/x]$, *if* $x \notin FV(v)$.

Proposition 1. *If* $t \to_m t'$ *in* HJ^+ *then either* $\overline{\overline{\rho}}(t) \to_{S_1} \overline{\overline{\rho}}(t')$ *or* $\overline{\overline{\rho}}(t) = \overline{\overline{\rho}}(t')$ *in* LJ^t.

The single anomaly is a *mid*-step of the form

$$(xl)\{x := v\} \to_m vl \ , \tag{1}$$

where $x \notin FV(l)$, which collapses in LJ^t because $\overline{\overline{\rho}}((xl)\{x := v\}) = \overline{\overline{\rho}}(vl)$. There is another (and last) anomaly, this time regarding head-cuts. The term $t[]$ is a S_1-redex and not a S_2-redex. We can split $\to_h$ as

(h_1) $t[] \to t$
(h_2) $tl \to insert(l, t)$, if t is not a λ-abstraction and $l \neq []$

and then:

Lemma 2. $\overline{\overline{\rho}}(insert(l, t)) = \overline{\overline{\rho}}(t)[(z)\overline{\overline{\rho}}(zl)/-]$, *if* $z \notin FV(l)$ *and* $l \neq []$.

Proposition 2. *If* $t \to_{h_i} t'$ *in* HJ^+ *then* $\overline{\overline{\rho}}(t) \to_{S_i} \overline{\overline{\rho}}(t')$ *in* LJ^t, $i = 1, 2$.

Finally:

Proposition 3. *If* $t \to_{log} t'$ *in* HJ^+ *then* $\overline{\overline{\rho}}(t) \to_{Log} \overline{\overline{\rho}}(t')$ *in* LJ^t.

Corollary 1. *The typable terms of* λ_H^+ *are strongly normalising.*

Proof. By strong normalisation of the tq-protocol, Propositions 1,2,3 and the fact that there can be no infinite reduction sequence starting from a λ_H^+-term and only made of steps (1). □

In the following it will be useful, namely for proving confluence of λ_H^+, to consider a translation $\overline{\overline{\varphi}} : LJ^t \to HJ^+$, as defined in Table 5.

Table 6. HJ and λ_H-calculus

$$u, v, t ::= xl \mid \lambda x.t \mid (\lambda x.t)(v :: l)$$
$$l, l' ::= [] \mid t :: l$$

$$Ax\frac{}{\Gamma; A \vdash [] : A} \quad Der\frac{\Gamma; A \vdash l : B}{\Gamma, x : A; - \vdash xl : B}$$

$$L\supset \frac{\Gamma; - \vdash t : A \quad \Gamma; B \vdash l : C}{\Gamma; A \supset B \vdash t :: l : C} \quad R\supset \frac{\Gamma, x : A; - \vdash t : B}{\Gamma; - \vdash \lambda x.t : A \supset B} x \notin \Gamma$$

$$beta - cut\,\frac{\Gamma, x : A; - \vdash t : B \quad \Gamma; - \vdash v : A \quad \Gamma; B \vdash l : C}{\Gamma; - \vdash (\lambda x.t)(v :: l) : C} x \notin \Gamma$$

$$(\beta_H) \quad (\lambda x.t)(v :: l) \to insert(l, (subst(v, x, t))$$

where

$$\begin{aligned}
subst(v, x, xl) &= insert(subst(v, x, l), v)\\
subst(v, x, yl) &= y\, subst(v, x, l),\ y \neq x\\
subst(v, x, \lambda y.t) &= \lambda y.subst(v, x, t)\\
subst(v, x, (\lambda y.t)(u :: l)) &= (\lambda y.subst(v, x, t))(subst(v, x, u) :: subst(v, x, l))\\
subst(v, x, u :: l) &= subst(v, x, u) :: subst(v, x, l)\\
subst(v, x, []) &= []
\end{aligned}$$

$$\begin{aligned}
insert([], t) &= t\\
insert(l, xl') &= x\, append(l', l),\ l \neq []\\
insert(l, \lambda x.t) &= (\lambda x.t)(u :: l'),\ l = u :: l'\\
insert(l, (\lambda x.t)(u :: l')) &= (\lambda x.t)\,(u :: append(l', l)),\ l \neq []
\end{aligned}$$

$$\begin{aligned}
append(t :: l, l') &= t :: append(l, l')\\
append([], l') &= l'
\end{aligned}$$

4 *HJ* and the λ_H-Calculus

HJ (see Table 6) was obtained by simplifying HJ^+ in such a way that the new calculus could be proved isomorphic to NJ by means of functions Ψ, Θ extending those defined in [2] between cut-free LJT and normal NJ.

The first thing to do is to get rid of mid-cuts and $\to_m$. This requires that *log* becomes

$$(\lambda x.t)(u :: l) \to subst(u, x, t)l \ . \tag{2}$$

However this is not enough. One problem is that we would have $\Theta(t[]) = \Theta(t)$ and thus Θ would not be injective. Hence we must require $l \neq []$ in every head-cut tl. The second problem is that $\Psi(M)$ will be h-normal, for all λ-term M. This requires two measures: (a) We restrict ourselves to h-normal terms. When mid-cuts are dropped, $t(u :: l)$ is h-normal iff t is a λ-abstraction. Thus head-cuts are required to be of the restricted form $(\lambda x.t)(u :: l)$. (b) We drop $\to_h$ and have to reduce immediately, by performing *insert*, the h-redexes generated in (2). Now $subst(u, x, t)l$ can itself be a h-redex and the h-redex ul' may be created at subformulas of t of the form xl'. This explains the first clause in the new definition of *subst* in HJ and the new version of (2) which we call β_H.

Every λ_H-term is a λ_H^+-term and next proposition says that β_H is a packet of several steps of reduction in HJ^+ and, indirectly, in the tq-protocol.

Proposition 4. *If $t \to_{\beta_H} t'$ in HJ then $t \to^+_{l,m,h} t'$ in HJ^+.*

Conversely, there is a translation $(_)^- : HJ^+ \to HJ$ defined by:

$$\begin{aligned}
(xl)^- &= xl^- \ , \\
(\lambda x.t)^- &= \lambda x.t^- \ , \\
(tl)^- &= insert(l^-, t^-) \ , \\
(t\{x := v\})^- &= subst(v^-, x, t^-) \ , \\
\\
(u :: l)^- &= u^- :: l^- \ , \\
([])^- &= [] \ .
\end{aligned}$$

Define $\overline{\rho}$ as the restriction of $\overline{\overline{\rho}}$ to HJ and $\overline{\varphi} = (_)^- \circ \overline{\overline{\varphi}}$. These $\overline{\rho}, \overline{\varphi}$ extend those of [3].

Proposition 5. $\overline{\varphi} \circ \overline{\rho} = id$.

Corollary 2. *The typable subcalculs of λ_H^+ is confluent.*

Proof. Since the typable subcalculus of λ_H^+ is strongly normalising, it suffices, by Newman's Lemma, to prove uniqueness of normal forms. Suppose $t \to^* t_1, t_2$ and that both t_i are cut-free. By the simulation results above, $\overline{\overline{\rho}}(t) \to^* \overline{\overline{\rho}}(t_1), \overline{\overline{\rho}}(t_2)$ and, since $\overline{\overline{\rho}}$ preserves cut-freeness, both $\overline{\overline{\rho}}(t_i)$ are cut-free. As $\overline{\overline{\rho}}$ preserves typability, $\overline{\overline{\rho}}(t_1)$ and $\overline{\overline{\rho}}(t_2)$ are obtained within the tq-protoc ol and, by confluence of the tq-protocol, $\overline{\overline{\rho}}(t_1) = \overline{\overline{\rho}}(t_2)$. Now crucially $t_1, t_2 \in HJ$ because t_1, t_2 are cut-free. Then, $t_1 = \overline{\varphi}(\overline{\rho}(t_1)) = \overline{\varphi}(\overline{\overline{\rho}}(t_1)) = \overline{\varphi}(\overline{\overline{\rho}}(t_2)) = \overline{\varphi}(\overline{\rho}(t_2)) = t_2$. □

Now let us turn to the relation between HJ and NJ. It is convenient to give the syntax of λ-calculus as

$$\begin{aligned}
M, N &::= x \mid \lambda x.M \mid app(A) \\
A &::= xM \mid (\lambda x.M)N \mid AM \ .
\end{aligned}$$

Translations Ψ and Θ between HJ and NJ are given in Table 7.

Table 7. Translations Ψ and Θ

$\Psi(x) = x[]$	$\Theta(x[]) = x$
$\Psi(\lambda x.M) = \lambda x.\Psi M$	$\Theta(x(u :: l)) = \Theta'(x\Theta u, l)$
$\Psi(app(A)) = \Psi'(A, [])$	$\Theta(\lambda x.t) = \lambda x.\Theta t$
	$\Theta((\lambda x.t)(u :: l)) = \Theta'((\lambda x.\Theta t)\Theta u, l)$
$\Psi'(xM, l) = x(\Psi M :: l)$	$=$
$\Psi'((\lambda x.M)N, l) = (\lambda x.\Psi M)(\Psi N :: l)$	$\Theta'(A, []) = app(A)$
$\Psi'(AM, l) = \Psi'(A, \Psi M :: l)$	$\Theta'(A, u :: l) = \Theta'(A\Theta u, l)$

Proposition 6. *$\Theta \circ \Psi = id$ and $\Psi \circ \Theta = id$.*

Proof. Extend the proof in [2]. □

Lemma 3. *$\Psi(M[N/x]) = subst(\Psi N, x, \Psi M)$.*

Corollary 3. *$\Theta(subst(v, x, t)) = \Theta t[\Theta v/x]$.*

The promised isomorphism of normalisation procedures is the following

Theorem 1.

1. *If $M \to_\beta M'$ in NJ then $\Psi M \to_{\beta_H} \Psi M'$ in HJ.*
2. *If $t \to_{\beta_H} t'$ in HJ then $\Theta t \to_\beta \Theta t'$ in NJ.*

Hence β and β_H are isomorphic, but β performs normalisation in NJ whereas β_H performs cut elimination in HJ.

5 Further Work

There are two main directions of further work.

First, to extend this work to the other connectives of intuitionistic predicate logic. Challenging seems to be the positive fragment and the treatment in this framework of the anomalies caused by disjunction and reported in [15].

Second, to generalise the whole enterprise to classical logic. The key players should be Herbelin's LKT and $\overline{\lambda}_\mu$-calculus [7], Parigot's natural deduction and λ_μ-calculus [8] and an appropriate LK^t. We plan to report on this in [4].

Acknowledgements

We thank Samson Abramsky for encouragement and guidance, Luís Pinto for his enthusiasm about this work and René Vestergaard for many chats on structural proof theory.

References

1. V. Danos, J-B. Joinet, and H. Schellinx. A new deconstructive logic: linear logic. *The Journal of Symbolic Logic*, 62(2):755–807, 1997.
2. R. Dyckhoff and L. Pinto. Cut-elimination and a permutation-free sequent calculus for intuitionistic logic. *Studia Logica*, 60:107–118, 1998.
3. R. Dyckhoff and L. Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoretical Computer Science*, 212:141–155, 1999.
4. J. Espírito Santo, 2000. PhD Thesis (in preparation).
5. J. Gallier. Constructive logics. Part I. *Theoretical Computer Science*, 110:248–339, 1993.
6. H. Herbelin. A λ-calculus structure isomorphic to a Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Proceedings of CSL'94*, volume 933 of *Lecture Notes in Computer Science*, pages 61–75. Springer-Verlag, 1995.
7. H. Herbelin. Sequents qu'on calcule, 1995. PhD Thesis, Université Paris VII.
8. M. Parigot. $\lambda\mu$-calculus: an algorithmic interpretation of classic natural deduction. In *Int. Conf. Logic Prog. Automated Reasoning*, volume 624 of *Lecture Notes in Computer Science*. Springer Verlag, 1992.
9. G. Pottinger. Normalization as a homomorphic image of cut-elimination. *Annals of Mathematical Logic*, 12:323–357, 1977.
10. D. Prawitz. *Natural Deduction. A Proof-Theoretical Study.* Almquist and Wiksell, Stockholm, 1965.
11. A.M. Ungar. *Normalization, Cut-eliminations and the Theory of Proofs.* Number 28 in CSLI Lecture Notes. 1992.
12. C. Urban and G. Bierman. Strong normalisation of cut-elimination in classical logic. In *Proceedings of TLCA'99*, Lecture Notes in Computer Science. Springer-Verlag, 1999.
13. R. Vestergaard and J.Wells. Cut rules and explicit substitutions. In *Second International Workshop on Explicit Substitutions*, 1999.
14. P. Wadler. A Curry-Howard isomorphism for sequent calculus, 1993. Manuscript.
15. J. Zucker. The correspondence between cut-elimination and normalization. *Annals of Mathematical Logic*, 7:1–112, 1974.

Negation Elimination from Simple Equational Formulae

Reinhard Pichler

Technische Universität Wien
reini@logic.at

Abstract. Equational formulae are first-order formulae over an alphabet $\mathcal{F}$ of function symbols whose only predicate symbol is syntactic equality. Unification problems are an important special case of equational formulae, where no universal quantifiers and no negation occur. By the *negation elimination problem* we mean the problem of deciding whether a given equational formula is semantically equivalent to a unification problem. This decision problem has many interesting applications in machine learning, logic programming, functional programming, constrained rewriting, etc. In this work we present a new algorithm for the negation elimination problem of equational formulae with purely existential quantifier prefix. Moreover, we prove the coNP-completeness for equational formulae in DNF and the Π_2^p-hardness in case of CNF.

1 Introduction

Equational formulae are first-order formulae over an alphabet $\mathcal{F}$ of function symbols whose only predicate symbol is syntactic equality. Unification problems are an important special case of equational formulae, where no universal quantifiers an no negation occur. By the *negation elimination problem* we mean the problem of deciding whether a given equational formula is semantically equivalent to a unification problem.

The so-called complement problems constitute a well-studied special case of the negation elimination problem, i.e.: given terms t, $s_1 \ldots, s_n$ over a signature $\mathcal{F}$, does there exist a finite set $\{r_1, \ldots, r_m\}$ of terms over $\mathcal{F}$, s.t. every $\mathcal{F}$-ground instance of t which is not an instance of any term s_i is an instance of some term r_j, and vice versa. Note that such terms $r_1, \ldots, r_m$ exist, iff negation elimination is possible from the equational formula $(\exists \boldsymbol{x})(\forall \boldsymbol{y})[z = t \wedge t \neq s_1 \wedge \ldots \wedge t \neq s_n]$, where the variables $\boldsymbol{x}$ in t and the variables $\boldsymbol{y}$ in the terms s_i are disjoint. Complement problems have many interesting applications in machine learning, logic programming, functional programming, etc (cf. [9]). In constrained rewriting, constraints are used to express certain rule application strategies (cf. [5]). Due to the failure of the critical pair lemma, one may eventually have to convert the constraints into equations only. Deciding whether equivalent equations exist, again corresponds to the negation elimination problem.

For the negation elimination problem in general (cf. [16], [3]) and for complement problems in particular (cf. [8]), several decision procedures have been

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 612–623, 2000.
© Springer-Verlag Berlin Heidelberg 2000

presented, which all have a rather high computational complexity. In [11], the coNP-hardness of negation elimination from complement problems was shown. This result was later extended in [10] to the coNP-completeness of these problems. The hardness proof in [11] also showed that negation elimination is at least as hard to decide as (un-)satisfiability. Hence, by the non-elementary complexity of the satisfiability problem (cf. [17]) we know that the negation elimination problem for arbitrary equational formulae is also non-elementary recursive.

Our main objective in this work is to device an efficient algorithm for the negation elimination problem of purely existentially quantified equational formulae. Note that these simple formulae are the target of the transformations given in [2] and [1] for equational formulae in arbitrary form. Hence, an efficient negation elimination algorithm for this special case is also important for the general case of negation elimination from arbitrary equational formulae. Moreover, we prove the coNP-completeness in case of DNF with purely existential quantifier prefix and the Π_2^p-hardness in case of CNF, respectively.

This paper is organized as follows: After recalling some basic notions in Sect. 2, we shall start our investigation of the negation elimination problem by considering existentially quantified conjunctions of equations and disequations in Sect. 3. In Sect. 4 and 5, this analysis will be extended to formulae in DNF and CNF, respectively. Finally, in Sect. 6 we give a short conclusion. Due to space limitations, proofs can only be sketched. The details are worked out in [14].

2 Preliminaries

An *equational formula* over an alphabet $\mathcal{F}$ of function symbols is a first-order formula with syntactic equality "=" as the only predicate symbol. Throughout this paper, we only consider the case where $\mathcal{F}$ is finite and contains at least one constant symbol (i.e.: a function symbol with arity 0) and at least one proper function symbol (i.e.: with arity greater than 0), since otherwise the negation elimination problem is trivial. A disequation $s \neq t$ is a short-hand notation for a negated equation $\neg(s = t)$. The trivially true formula is denoted by $\top$ and the trivially false one by $\bot$. An interpretation is given through a ground substitution σ over $\mathcal{F}$, whose domain coincides with the free variables of the equational formula. The trivial formula $\top$ evaluates to true in every interpretation. Likewise, $\bot$ always evaluates to false. A single equation $s = t$ is validated by a ground substitution σ, if $s\sigma$ and $t\sigma$ are syntactically identical. The connectives $\wedge$, $\vee$, $\neg$, $\exists$ and $\forall$ are interpreted as usual. A ground substitution σ which validates an equational formula $\mathcal{P}$ is called a *solution* of $\mathcal{P}$. In order to distinguish between syntactical identity and the semantic equivalence of two equational formulae, we shall use the notation "$\equiv$" and "$\approx$", respectively, i.e.: $\mathcal{P} \equiv \mathcal{Q}$ means that the two formulae $\mathcal{P}$ and $\mathcal{Q}$ are *syntactically identical*, while $\mathcal{P} \approx \mathcal{Q}$ means that the two formulae are *semantically equivalent* (i.e.: they have the same set of solutions). Moreover, by $\mathcal{P} \leq \mathcal{Q}$ we denote that all solutions of $\mathcal{P}$ are also solutions of $\mathcal{Q}$. We shall sometimes use term tuples as a short-hand notation for a disjunction of disequations or a conjunction of equations, respectively, i.e.: For term tuples

$\boldsymbol{s} = (s_1, \ldots, s_k)$ and $\boldsymbol{t} = (t_1, \ldots, t_k)$, we shall abbreviate "$s_1 = t_1 \wedge \ldots \wedge s_k = t_k$" and "$s_1 \neq t_1 \vee \ldots \vee s_k \neq t_k$" to "$\boldsymbol{s} = \boldsymbol{t}$" and "$\boldsymbol{s} \neq \boldsymbol{t}$", respectively. Note that conjunctions of equations and disequations can be easily simplified via unification, i.e.: Let $\vartheta = \{z_1 \leftarrow r_1, \ldots, z_n \leftarrow r_n\}$ be the mgu of $\boldsymbol{s}$ and $\boldsymbol{t}$. Then $\boldsymbol{s} = \boldsymbol{t}$ is equivalent to $z_1 = r_1 \wedge \ldots \wedge z_n = r_n$. Likewise, $\boldsymbol{s} \neq \boldsymbol{t}$ is equivalent to $z_1 \neq r_1 \vee \ldots \vee z_n \neq r_n$. As a short-hand notation, we shall write E$qu(\vartheta)$ and D$isequ(\vartheta)$, for these simplified equations and disequations, respectively.

An *implicit generalization* over a signature $\mathcal{F}$ is a construct of the form $I = \boldsymbol{t}/\boldsymbol{t}_1 \vee \ldots \vee \boldsymbol{t}_m$ with the intended meaning that I represents all ground term tuples over $\mathcal{F}$, that are instances of $\boldsymbol{t}$ but not of any tuple $\boldsymbol{t}_i$ (cf. [8]). In contrast, an *explicit generalization* is a disjunction $\boldsymbol{r}_1 \vee \ldots \vee \boldsymbol{r}_m$ of term tuples over $\mathcal{F}$, which contains all ground term tuples over $\mathcal{F}$, that are an instance of at least one tuple $\boldsymbol{r}_i$. A ground term tuple $\boldsymbol{s} = (s_1, \ldots, s_k)$ is an instance of the implicit generalization $I = \boldsymbol{t}/\boldsymbol{t}_1 \vee \ldots \vee \boldsymbol{t}_m$, iff the ground substitution $\sigma = \{z_1 \leftarrow s_1, \ldots, z_k \leftarrow s_k\}$ is a solution of the equational formula $\mathcal{P} \equiv (\exists \boldsymbol{x})(\forall \boldsymbol{y})\,[\boldsymbol{z} = \boldsymbol{t} \wedge \boldsymbol{t} \neq \boldsymbol{s}_1 \wedge \ldots \wedge \boldsymbol{t} \neq \boldsymbol{s}_n]$, where the variables $\boldsymbol{x}$ occurring in $\boldsymbol{t}$ and the variables $\boldsymbol{y}$ occurring in the tuples $\boldsymbol{t}_i$ are disjoint. Moreover, the question as to whether the ground term tuples contained in I can be represented by an explicit generalization is equivalent to the negation elimination problem of $\mathcal{P}$.

A term t is called *linear*, iff it has no multiple variable occurrences. Moreover, a term $t\vartheta$ is a *linear instance* of t, iff all terms in the range r$g(\vartheta)$ are linear and for all variables x,y in V$ar(t)$ with $x \neq y$, $x\vartheta$ and $y\vartheta$ have no variables in common. By the domain closure axiom, every ground term over $\mathcal{F}$ is an instance of the disjunction $\bigvee_{f \in \mathcal{F}} f(x_1, \ldots, x_{\alpha(f)})$, where the x_i's are pairwise distinct variables and $\alpha(f)$ denotes the arity of f. In [8], this fact is used to provide a representation of the complement of a *linear* instance $\boldsymbol{t}\vartheta$ of $\boldsymbol{t}$ (i.e.: of all ground term tuples that are contained in $\boldsymbol{t}$ but not in $\boldsymbol{t}\vartheta$). If every term tuple $\boldsymbol{t}_i$ on the right-hand side of an implicit generalization $I = \boldsymbol{t}/\boldsymbol{t}_1 \vee \ldots \vee \boldsymbol{t}_m$ is a linear instance of $\boldsymbol{t}$, then this representation of the complement immediately yields an equivalent explicit representation E of I, i.e.: Let $P_i = \{\boldsymbol{p}_{i1}, \ldots, \boldsymbol{p}_{in_i}\}$ denote the complement of $\boldsymbol{t}_i$ w.r.t. t and suppose that the terms $\boldsymbol{p}_{ij}$ are pairwise variable disjoint. Then I is equivalent to $E = \bigvee_{j_1=1}^{n_1} \ldots \bigvee_{j_m=1}^{n_m} \mathrm{m}gi(\boldsymbol{p}_{1j_1}, \ldots, \boldsymbol{p}_{mj_m})$, where m$gi$ denotes the most general instance (cf. [11], Proposition 3.4 and Corollary 3.5).

In [4], a representation of the complement of an instance $\boldsymbol{t}\vartheta$ w.r.t. $\boldsymbol{t}$ is provided also for the case where $\boldsymbol{t}\vartheta$ is not necessarily linear. The idea of this representation is to construct the instances $\boldsymbol{t}\sigma$ of $\boldsymbol{t}$, which are not contained in $\boldsymbol{t}\vartheta$, in the following way: Consider the tree representation of ϑ, "deviate" from this representation at some node and close all other branches of σ as early as possible with new, pairwise distinct variables. Depending on the label of a node, this deviation can be done in two different ways: If a node is labelled by a function symbol from $\mathcal{F}$ (note that constants are considered as function symbols of arity 0), then this node has to be labelled by a different function symbol from $\mathcal{F}$. If a node is labelled by a variable which also occurs at some other position, then the two occurrences of this variable have to be replaced by two fresh variables x, y and the constraint $x \neq y$ has to be added. However, if a node is labelled by

a variable which occurs nowhere else, then no deviation at all is possible at this node. For our purposes here it suffices to know that the complement of a term tuple $\boldsymbol{t}\vartheta$ w.r.t. $\boldsymbol{t}$ can be represented by a finite set of constrained term tuples $P = \{[\boldsymbol{p}_1 : X_1], \ldots, [\boldsymbol{p}_n : X_n]\}$, where $[\boldsymbol{p}_i : X_i]$ denotes the set of all ground instances $\boldsymbol{p}_i\sigma$ of $\boldsymbol{p}_i$, s.t. σ is a solution of the equational formula X_i. Moreover, the number of elements in P is quadratically bounded and the size of each constrained term tuple $[\boldsymbol{p}_i : X_i]$ is linearly bounded w.r.t. the size of the tuples $\boldsymbol{t}$ and $\boldsymbol{t}\vartheta$. Finally, the constraints X_i are either disequations or the trivially true formula $\top$. Note that this representation of the complement can be easily extended to implicit generalizations, namely: Let $I = \boldsymbol{t}/\boldsymbol{t}\vartheta_1 \vee \ldots \vee \boldsymbol{t}\vartheta_m$ be an implicit generalization. Then the complement of I (i.e. the set of ground term tuples over $\mathcal{F}$ which are not contained in I) can be represented by $P \cup \{\boldsymbol{t}\vartheta_1, \ldots, \boldsymbol{t}\vartheta_m\}$, where P is a representation of the complement of $\boldsymbol{t}$.

Let $S = \{s_1, \ldots, s_n\}$ be a a set of terms over $\mathcal{F}$. Then the set of all common ground terms of these terms can be computed via unification, namely: Let $s_1, \ldots, s_n$ be pairwise variable disjoint and let $\mu = \mathrm{m}gu(s_1, \ldots, s_n)$ denote the most general unifier of these terms. Then $s_1\mu$ contains exactly those ground terms over $\mathcal{F}$, which are contained in all terms s_i. Similarly, the common ground instances of constrained term tuples $[\boldsymbol{p}_1 : X_1], \ldots, [\boldsymbol{p}_n : X_n]$ can be represented by $[\boldsymbol{p}_1\mu : Z\mu]$, where $\mu = \mathrm{m}gu(\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n)$ denotes the most general unifier and $Z \equiv X_1 \wedge \ldots \wedge X_n$. Moreover, if the constraints X_i are either disequations or of the form $\top$, then such an intersection is non-empty, iff the mgu μ exists and $Z\mu$ contains no trivial disequation of the form $t \neq t$ (cf. [1], Lemma 2).

3 Conjunctions of Equations and Disequations

Recall that the satisfiability problem of (existentially quantified) conjunctions of equations and disequations can be easily decided via unification, namely: Let $\mathcal{P} \equiv (\exists \boldsymbol{x})(e_1 \wedge \ldots \wedge e_k \wedge d_1 \wedge \ldots \wedge d_l)$, where the e_i's are equations and the d_i's are disequations. Then $\mathcal{P}$ is satisfiable, iff the equations are unifiable and the application of the most general unifier $\mu = \mathrm{m}gu(e_1, \ldots, e_k)$ to the disequations does not produce a trivial disequation $d_i\mu$ of the form $t \neq t$. In this section we show that also for the negation elimination problem of such formulae, there is an efficient decision procedure based on unification. To this end, we provide simplifications of the equations and of the disequations in the Lemmas 3.1 and 3.2, respectively. In Theorem 3.1 we shall prove that these simplifications are indeed all we need for deciding the negation elimination problem.

Lemma 3.1. (simplification of the equations) *Let $\mathcal{P} \equiv (\exists \boldsymbol{x})(e_1 \wedge \ldots \wedge e_k \wedge d_1 \wedge \ldots \wedge d_l)$ be an equational formula over $\mathcal{F}$, where the e_i's are equations and the d_i's are disequations. Moreover, let $\boldsymbol{z} = (z_1, \ldots, z_n)$ denote the free variables occurring in $\mathcal{P}$. Then $\mathcal{P}$ may be transformed in the following way:*

<u>case 1</u>: If the equations $e_1, \ldots, e_k$ are not unifiable, or if $e_1, \ldots, e_k$ are unifiable with $\mu = \mathrm{m}gu(e_1, \ldots, e_k)$, s.t. at least one disequation $d_i\mu$ is trivially false (i.e.: it is of the form $t \neq t$ for some term t), then $\mathcal{P}$ is equivalent to $\mathcal{P}' \equiv \bot$.

<u>case 2</u>: If the equations $e_1, \ldots, e_k$ are unifiable with $\mu = \mathrm{mgu}(e_1, \ldots, e_k)$ and none of the disequations $d_i\mu$ is trivially false, then we define $\mathcal{P}'$ as follows: W.l.o.g. we assume that $\mathrm{dom}(\mu) = \{z_1, \ldots, z_\alpha\}$ for some $\alpha \leq n$. Now let $\boldsymbol{u} = (u_{\alpha+1}, \ldots, u_n)$ be a vector of fresh, pairwise distinct variables and let $\nu = \{z_{\alpha+1} \leftarrow u_{\alpha+1}, \ldots, z_n \leftarrow u_n\}$. Then $\mathcal{P}$ is equivalent to $\mathcal{P}' \equiv (\exists \boldsymbol{x})(\exists \boldsymbol{u})\big(\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=1}^{l} d_i\mu\nu\big)$ with $\boldsymbol{t} = (z_1\mu\nu, \ldots, z_\alpha\mu\nu, u_{\alpha+1}, \ldots, u_n)$.

Proof. (Sketch): Case 1 corresponds to the satisfiability test mentioned above. The transformation in case 2 consists of several sub-steps, namely: Let $\boldsymbol{x} = (x_1, \ldots, x_m\}$. W.l.o.g. we assume that the mgu μ is of the form $\mu = \{z_1 \leftarrow t_1, \ldots, z_\alpha \leftarrow t_\alpha, x_1 \leftarrow s_1, \ldots, x_\beta \leftarrow s_\beta\}$ for some $\beta \leq m$. Then, by the definition of mgu's, $\mathcal{P}$ is equivalent to $\mathcal{R} \equiv (\exists \boldsymbol{x})\big[\bigwedge_{i=1}^{\alpha}(z_i = t_i) \wedge \bigwedge_{i=1}^{\beta}(x_i = s_i) \wedge \bigwedge_{i=1}^{l} d_i\mu\big]$. Moreover, for any equational formula $\mathcal{Q}$ and any variable u not occurring in $\mathcal{Q}$, the equivalence $\mathcal{Q} \approx (\exists u)[z = u \wedge \mathcal{Q}\{z \leftarrow u\}]$ holds. Hence, $\mathcal{R}$ is equivalent to $\mathcal{R}' \equiv (\exists \boldsymbol{x})(\exists \boldsymbol{u})\big[\bigwedge_{i=\alpha+1}^{n}(z_i = u_i) \wedge \bigwedge_{i=1}^{\alpha}(z_i = t_i\nu) \wedge \bigwedge_{i=1}^{\beta}(x_i = s_i\nu) \wedge \bigwedge_{i=1}^{l} d_i\mu\nu\big]$. Finally, note that the x_i's are existentially quantified variables which occur nowhere else in $\mathcal{R}'$. Hence, it can be easily shown that no solutions are added to $\mathcal{R}'$, if we delete the equations $x_i = s_i\nu$ (for details, see [14]). □

Lemma 3.2. (simplification of the disequations) *Let $\mathcal{P} \equiv (\exists \boldsymbol{x})\big(\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=1}^{l} d_i\big)$ be an equational formula that results from the transformation according to case 2 from Lemma 3.1 above (i.e., in particular, the free variables $\boldsymbol{z}$ of $\mathcal{P}$ neither occur in $\boldsymbol{t}$ nor in the disequations). Then, every disequation d_i can be further transformed as follows:*

<u>case 1</u>: If the equation $\neg d_i$ is not unifiable, then d_i is equivalent to $\top$ and may therefore be deleted.

<u>case 2</u>: Otherwise, let $\mathrm{mgu}(\neg d_i) = \vartheta_i = \{v_1 \leftarrow s_1, \ldots, v_\gamma \leftarrow s_\gamma\}$.

<u>case 2.1</u>: If $\mathrm{dom}(\vartheta_i) \cup \mathrm{Var}(\mathrm{rg}(\vartheta_i)) \subseteq \mathrm{Var}(\boldsymbol{t})$ (i.e.: the v_j's and all variables in the terms s_j also occur in $\boldsymbol{t}$), then we replace d_i by $\mathrm{Disequ}(\vartheta_i) \equiv \bigvee_{j=1}^{\gamma} v_j \neq s_j$.

<u>case 2.2</u>: If $\mathrm{dom}(\vartheta_i) \cup \mathrm{Var}(\mathrm{rg}(\vartheta_i)) \not\subseteq \mathrm{Var}(\boldsymbol{t})$, then d_i may be deleted.

Proof. (Sketch): The correctness of the cases 1 and 2.1 is clear by the definition of mgu's. W.l.o.g. we assume that the disequations $d_1, \ldots, d_\delta$ for some $\delta \leq l$ may be deleted via case 2.2. Moreover, let $\boldsymbol{y} \subseteq \boldsymbol{x}$ denote those variables from $\boldsymbol{x}$ which do not occur in $\boldsymbol{t}$ and, for every $i \in \{1, \ldots, \delta\}$, let $v_{j_i} \neq s_{j_i}$ denote a disequation in $\mathrm{Disequ}(\vartheta_i)$ which contains a variable from $\boldsymbol{y}$. Then the equivalence $(\exists \boldsymbol{y})\mathcal{Q} \wedge \bigwedge_{i=1}^{\delta} v_{j_i} \neq s_{j_i} \approx \mathcal{Q}$ holds for any equational formula $\mathcal{Q}$ that contains no variable from $\boldsymbol{y}$. Moreover, $v_{j_i} \neq s_{j_i} \leq d_i$ holds for every $i \in \{1, \ldots, \delta\}$. We thus have $\mathcal{P} \equiv (\exists \boldsymbol{x})\big(\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=1}^{l} d_i\big) \leq (\exists \boldsymbol{x})\big(\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=\delta+1}^{l} d_i\big) \approx (\exists \boldsymbol{x})\big(\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=1}^{\delta} v_{j_i} \neq s_{j_i} \wedge \bigwedge_{i=\delta+1}^{l} d_i\big) \leq \mathcal{P}$. Hence, $\mathcal{P}$ and $(\exists \boldsymbol{x})\big(\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=\delta+1}^{l} d_i\big)$ are indeed equivalent. □

An equational formula of the form $\mathcal{P} \equiv (\exists \boldsymbol{x})\big[\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=1}^{l} \mathrm{Disequ}(\vartheta_i)\big]$ corresponds to the implicit generalization $I = \boldsymbol{t}/(\boldsymbol{t}\vartheta_1 \vee \ldots \vee \boldsymbol{t}\vartheta_l)$ (cf. Sect. 2). In

particular, negation elimination from $\mathcal{P}$ corresponds to the conversion of I into an explicit generalization. Hence, one way to decide the negation elimination problem for formulae resulting from the simplifications of the Lemmas 3.1 and 3.2 is to apply the algorithm from [8] (which has an exponential worst case complexity) to the corresponding implicit generalization. In the following theorem we claim that (due to the special form of the disequations), deciding the negation elimination problem for such formulae is in fact much cheaper than this.

Theorem 3.1. (negation elimination from simplified conjunctions) *Let* $\mathcal{P} \equiv (\exists \boldsymbol{x})\big[\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=1}^{l} \mathrm{D}isequ(\vartheta_i)\big]$ *be an equational formula which results from the transformation of Lemma 3.1 followed by Lemma 3.2. Then negation elimination from* $\mathcal{P}$ *is possible, iff for all* $i \in \{1, \ldots, l\}$, $\mathrm{V}ar(\mathrm{r}g(\vartheta_i)) = \emptyset$ *holds.*

Proof. (Sketch): The "if"-direction follows immediately from the correspondence between the equational formula $\mathcal{P} \equiv (\exists \boldsymbol{x})\big[\boldsymbol{z} = \boldsymbol{t} \wedge \bigwedge_{i=1}^{l} \mathrm{D}isequ(\vartheta_i)\big]$ and the implicit generalization $I = \boldsymbol{t}/(\boldsymbol{t}\vartheta_1 \vee \ldots \vee \boldsymbol{t}\vartheta_l)$. In particular, a disequation $\mathrm{D}isequ(\vartheta_i)$ for which $\mathrm{V}ar(\mathrm{r}g(\vartheta_i)) = \emptyset$ holds, corresponds to a linear instance $\boldsymbol{t}\vartheta_i$ of $\boldsymbol{t}$. For the "only if"-direction, it is shown in [14] that the implicit generalization I corresponding to $\mathcal{P}$ can be split into disjoint generalizations $I_1, I_2, \ldots$ via the complement of the linear term tuples $\boldsymbol{t}\vartheta_i$ on the right-hand side of I, s.t. there is at least one implicit generalization $I_j = \boldsymbol{s}_j/(\boldsymbol{s}_j\eta_{j1} \vee \ldots \vee \boldsymbol{s}_j\eta_{jl_j})$ where all term tuples $\boldsymbol{s}_j\eta_{jk}$ are non-linear instances of $\boldsymbol{s}_j$. By Proposition 4.6 from [8], we may then conclude that negation elimination is impossible for the equational formula corresponding to I_j and, therefore, also for $\mathcal{P}$. □

Note that the transformations in the Lemmas 3.1 and 3.2 above as well as testing the condition from Theorem 3.1 can of course be done in polynomial time, provided that terms are represented as directed, acyclic graphs (cf. [12]). In [11], a similar result was proven for the negation elimination from quantifier-free conjunctions of equations and disequations. To this end, the notion of "effectively ground" disequations was introduced, i.e.: Let d be a disequation and let μ denote the mgu of the equations. Then d is *effectively ground*, iff the mgu λ of $\neg d\mu$ is a ground substitution. A disequation d, which may be deleted without changing the meaning of the equational formula, is called *redundant*. In particular, d is redundant, if $\neg d\mu$ is not unifiable. Then it is shown in [11], that negation elimination from a quantifier-free conjunction of equations and disequations is possible, iff every non-redundant disequation is effectively ground. In other words, by the transformations from the Lemmas 3.1 and 3.2, we have extended the notion of "effectively ground" disequations to the case where existentially quantified variables are allowed. In the following example, we put these transformations together with Theorem 3.1 to work:

Example 3.1. Let $\mathcal{P} \equiv (\exists x_1, x_2, x_3)\big[g(z_1, x_2) = g(f(x_1), f(x_3)) \wedge z_2 \neq z_1 \wedge f(z_2) \neq f(a) \wedge x_3 \neq z_1\big]$ be an equational formula over $\mathcal{F} = \{a, f, g\}$. Then the m$gu$ μ of the equations has the form $\mu = \{z_1 \leftarrow f(x_1), x_2 \leftarrow f(x_3)\}$ and, therefore, $\mathcal{P}$ is equivalent to $\mathcal{P}' \equiv (\exists x_1, x_2, x_3)\big[(z_1, x_2) = (f(x_1), f(x_3)) \wedge z_2 \neq f(x_1) \wedge f(z_2) \neq f(a) \wedge x_3 \neq f(x_1)\big]$. In order to bring the free variable z_2 to the

left-hand side of the equations, we define the substitution $\nu = \{z_2 \leftarrow u_2\}$ according to Lemma 3.1. Then $\mathcal{P}'$ is equivalent to $\mathcal{P}'' \equiv (\exists x_1, x_2, x_3, u_2)\big[(z_1, z_2, x_2) = (f(x_1), u_2, f(x_3)) \wedge u_2 \neq f(x_1) \wedge f(u_2) \neq f(a) \wedge x_3 \neq f(x_1)\big]$. Finally, by Lemma 3.1, we may delete the equation $x_2 = f(x_3)$. We thus get $\mathcal{P}''' \equiv (\exists x_1, x_2, x_3, u_2)$ $\big[(z_1, z_2) = (f(x_1), u_2) \wedge u_2 \neq f(x_1) \wedge f(u_2) \neq f(a) \wedge x_3 \neq f(x_1)\big]$.

By Lemma 3.2, we may transform the disequations in the following way: $f(u_2) \neq f(a)$ may be simplified to $u_2 \neq a$. $x_3 \neq f(x_1)$ may be deleted (due to the presence of the variable x_3, which does not occur any more in the equations). Finally, $u_2 \neq f(x_1)$ is left unchanged. Hence, the original formula $\mathcal{P}$ is equivalent to $\mathcal{R} \equiv (\exists x_1, x_2, u_2)\big[(z_1, z_2) = (f(x_1), u_2) \wedge u_2 \neq f(x_1) \wedge u_2 \neq a\big]$. By Theorem 3.1, negation elimination from $\mathcal{R}$ (and, therefore, also from $\mathcal{P}$) is impossible, since the disequation $u_2 \neq f(x_1)$ in $\mathcal{R}$ is based on a non-ground substitution.

4 Equational Formulae in DNF

The algorithms in [2] and [1] for solving equational formulae result in the transformation of an arbitrary equational formula into a so-called "definition with constraints", which is basically an existentially quantified equational formula in DNF. As far as the satisfiability of equational formulae is concerned, such a transformation is indeed all we need. Note that the equational formula $\mathcal{D} \equiv (\exists \boldsymbol{x})\mathcal{D}_1 \vee \ldots \vee \mathcal{D}_n$, where the $\mathcal{D}_i$'s are conjunctions of equations and disequations, is satisfiable, iff at least one of the subformulae $(\exists \boldsymbol{x})\mathcal{D}_i$ is satisfiable. As has already been pointed out in Sect. 3, the latter condition can be tested very efficiently via unification. In this section we extend the simplifications for conjunctions of equations and disequations from the previous section to equational formulae in DNF. In Lemma 4.1 below, we provide a transformation which may be applied to disequations of the form $\mathrm{D}isequ(\vartheta_{ij})$ with $\mathrm{V}ar(\mathrm{r}g(\vartheta_{ij})) \neq \emptyset$. In Theorem 4.1 we shall then show that either this transformation allows us to eliminate all disequations that are based on non-ground substitutions, or negation elimination is impossible. Unfortunately, this algorithm has exponential complexity. However, by the coNP-completeness shown in Theorem 4.2, we can hardly expect to find a significantly better algorithm.

Lemma 4.1. (simplification of a DNF) *Let $\mathcal{P}$ be an equational formula with $\mathcal{P} \equiv (\exists \boldsymbol{x})\big[\boldsymbol{z} = \boldsymbol{t}_1 \wedge \bigwedge_{j=1}^{l_1} \mathrm{D}isequ(\vartheta_{1j})\big] \vee \ldots \vee \big[\boldsymbol{z} = \boldsymbol{t}_n \wedge \bigwedge_{j=1}^{l_n} \mathrm{D}isequ(\vartheta_{nj})\big]$, s.t. each disjunct of $\mathcal{P}$ has already been simplified via Lemma 3.1 followed by Lemma 3.2. Let $I_i = \boldsymbol{t}_i/(\boldsymbol{t}_i\vartheta_{i1} \vee \ldots \vee \boldsymbol{t}_i\vartheta_{il_i})$ be the implicit generalization corresponding to the i-th disjunct of $\mathcal{P}$ and let $P_i = \{[\boldsymbol{p}_{i1} : X_{i1}], \ldots, [\boldsymbol{p}_{im_i} : X_{im_i}]\}$ denote the complement of I_i. Moreover, suppose that all term tuples $\boldsymbol{p}_{j\alpha}$ and $\boldsymbol{t}_i\vartheta_{i\beta}$ are pairwise variable disjoint. Finally, for every ϑ_{ij} with $\mathrm{V}ar(\mathrm{r}g(\vartheta_{ij})) \neq \emptyset$, we define $\Lambda(i,j)$ as the following set of substitutions:*

$$\begin{aligned}\Lambda(i,j) = \{\, \lambda \mid (\exists \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n),\ \textit{s.t.}\\ \mu = \mathrm{m}gu(\boldsymbol{t}_i\vartheta_{ij}, \boldsymbol{p}_{1\alpha_1}, \ldots, \boldsymbol{p}_{(i-1)\alpha_{i-1}}, \boldsymbol{p}_{(i+1)\alpha_{i+1}}, \ldots, \boldsymbol{p}_{n\alpha_n})\ \textit{exists}\\ \textit{and } (X_{1\alpha_1} \wedge \ldots \wedge X_{(i-1)\alpha_{i-1}} \wedge X_{(i+1)\alpha_{i+1}} \wedge \ldots \wedge X_{n\alpha_n})\mu \textit{ contains}\\ \textit{no trivial disequation and } \lambda = \mu|_{\mathrm{V}ar(\mathrm{r}g(\vartheta_{ij}))}\,\}\end{aligned}$$

If $\Lambda(i,j)$ contains only ground substitutions, then the disequation $\mathrm{D}isequ(\vartheta_{ij})$ *may be replaced by the conjunction* $\bigwedge_{\lambda\in\Lambda(i,j)} \mathrm{D}isequ(\vartheta_{ij}\lambda)$ *of disequations.*

Proof. (Sketch): We make again use of the correspondence between implicit generalizations and equational formulae. In particular, replacing the disequation $\mathrm{D}isequ(\vartheta_{ij})$ by the conjunction $\bigwedge_{\lambda\in\Lambda(i,j)} \mathrm{D}isequ(\vartheta_{ij}\lambda)$ corresponds to the replacement of the term tuple $\boldsymbol{t}_i\vartheta_{ij}$ on the right-hand side of I_i by the disjunction of term tuples $\bigvee_{\lambda\in\Lambda(i,j)} \boldsymbol{t}_i\vartheta_{ij}\lambda$. Now let $\mathcal{I} = I_1 \vee \ldots \vee I_n$ denote the disjunction of implicit generalizations corresponding to $\mathcal{P}$ and let $\mathcal{I}' = I'_1 \vee \ldots I'_n$ denote the disjunction of implicit generalizations corresponding to the formula $\mathcal{P}'$ which results from a simultaneous application of the above replacement rule to all disequations $\mathrm{D}isequ(\vartheta_{ij})$. We have to show that then $\mathcal{I}$ and $\mathcal{I}'$ are equivalent. Actually, $\mathcal{I} \subseteq \mathcal{I}'$ clearly holds, since every tuple $\boldsymbol{t}_i\vartheta_{ij}\lambda$ is an instance of $\boldsymbol{t}_i\vartheta_{ij}$ and, therefore, $I_i \subseteq I'_i$ trivially holds for every $i \in \{1,\ldots,n\}$. In order to see that $\mathcal{I}' \subseteq \mathcal{I}$ holds as well, note that every implicit generalization I'_i is basically obtained from I_i by restricting the terms $\boldsymbol{t}_i\vartheta_{ij}$ to those instances, which are *not* contained in the remaining implicit generalizations I_j. The correctness of this restriction is due to the relation $[A - B] \cup C = [A - (B - C)] \cup C$, which holds for arbitrary sets A, B and C. □

The transformation from Lemma 4.1 is illustrated in the following example:

Example 4.1. Let $\mathcal{F} = \{a, g\}$ and let $\mathcal{P} \equiv (\exists x_1, x_2, x_3)(\mathcal{D}_1 \vee \mathcal{D}_2 \vee \mathcal{D}_3)$ with

$$\begin{aligned}
&\mathcal{D}_1 \equiv (z_1, z_2, z_3) = (x_1, x_2, g(x_1)) \wedge x_1 \neq x_2\\
&\mathcal{D}_2 \equiv (z_1, z_2, z_3) = (g(x_1), g(x_2), x_3) \wedge x_2 \neq a\\
&\mathcal{D}_3 \equiv (z_1, z_2, z_3) = (g(x_1), x_2, x_3) \wedge x_3 \neq g^2(x_1),
\end{aligned}$$

In order to transform the disjuncts $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ via Lemma 4.1, we consider the corresponding implicit generalizations I_1, I_2 and I_3:

$$\begin{aligned}
&I_1 = (x_1, x_2, g(x_1))/(x_2, x_2, g(x_2))\\
&I_2 = (g(x_1), g(x_2), x_3)/(g(x_1), g(a), x_3)\\
&I_3 = (g(x_1), x_2, x_3)/(g(x_1), x_2, g^2(x_1))
\end{aligned}$$

These implicit generalizations have the following complement representations P_1, P_2 and P_3, respectively (note that we may omit the constraint "$\top$" from tuples of the form $[t : \top]$. Moreover, we rename the variables apart:

$$\begin{aligned}
&P_1 = \{(y_{11}, y_{12}, a), [(y_{11}, y_{12}, g(y_{13}) : y_{11} \neq y_{13}], (y_{11}, y_{11}, g(y_{11}))\}\\
&P_2 = \{(a, y_{21}, y_{22}), (y_{21}, a, y_{22}), (g(y_{21}), g(a), y_{22})\}\\
&P_3 = \{(a, y_{31}, y_{32}), (g(y_{31}), y_{32}, g^2(y_{31}))\}
\end{aligned}$$

The disequation $x_2 \neq a$ in $\mathcal{D}_2$ cannot be further simplified, since $\vartheta_{21} = \{x_2 \leftarrow a\}$ already is a ground substitution. So we only have to transform the disequations $x_1 \neq x_2$ in $\mathcal{D}_1$ and $x_3 \neq g^2(x_1)$ in $\mathcal{D}_3$, respectively. The set $\Lambda(1,1)$ of substitutions which have to be applied to $x_1 \neq x_2$ in $\mathcal{D}_1$ can be computed as follows. By μ_{α_2,α_3} we denote the m*gu* of $(x_2, x_2, g(x_2))$ with the α_2-th term tuple from P_2 and the α_3-th term tuple from P_3. Likewise, by $\lambda_{\alpha_2,\alpha_3}$ we denote the restriction of μ_{α_2,α_3} to the variable x_2, which is the only variable occurring in the range of $\vartheta_{11} = \{x_1 \leftarrow x_2\}$.

$$\begin{aligned}
\mu_{1,1} &= \mathrm{m}gu\big((x_2, x_2, g(x_2)), (a, y_{21}, y_{22}), (a, y_{31}, y_{32})\big) = \\
&\quad \{x_2 \leftarrow a, y_{21} \leftarrow a, y_{22} \leftarrow g(a), y_{31} \leftarrow a, y_{32} \leftarrow g(a)\} \Rightarrow \lambda_{1,1} = \{x_2 \leftarrow a\} \\
\mu_{2,1} &= \mathrm{m}gu\big((x_2, x_2, g(x_2)), (y_{21}, a, y_{22}), (a, y_{31}, y_{32})\big) \qquad \Rightarrow \lambda_{2,1} = \{x_2 \leftarrow a\} \\
\mu_{3,2} &= \mathrm{m}gu\big((x_2, x_2, g(x_2)), (g(y_{21}), g(a), y_{22}), (g(y_{31}), y_{32}, g^2(y_{31}))\big) \\
&\quad \Rightarrow \lambda_{3,2} = \{x_2 \leftarrow g(a)\}
\end{aligned}$$

Hence, by Lemma 4.1, the disequation $\mathrm{D}isequ(\vartheta_{11}) \equiv x_1 \neq x_2$ in $\mathcal{D}_1$ may be replaced by the conjunction $\mathrm{D}isequ(\vartheta_{11} \circ \{x_2 \leftarrow a\}) \wedge \mathrm{D}isequ(\vartheta_{11} \circ \{x_2 \leftarrow g(a)\})$, i.e.: $\mathcal{D}_1$ may be transformed into $\mathcal{D}_1' \equiv (z_1, z_2, z_3) = (x_1, x_2, g(x_1)) \wedge (x_1, x_2) \neq (a, a) \wedge (x_1, x_2) \neq (g(a), g(a))$.

Now we compute the set $\Lambda(3, 1)$ of substitutions which have to be applied to the disequation $x_3 \neq g^2(x_1)$ in $\mathcal{D}_3$. Again we use the notation μ_{α_1,α_2} to denote the mgu of $(g(x_1), x_2, g^2(x_1))$ with the α_1-th term tuple from P_1 and the α_2-th term tuple from P_2. Likewise, by $\lambda_{\alpha_1,\alpha_2}$ we denote the restriction of μ_{α_1,α_2} to the variable x_1.

$$\begin{aligned}
\mu_{2,2} &= \mathrm{m}gu\big((g(x_1), x_2, g^2(x_1)), (y_{11}, y_{12}, g(y_{13})), (y_{21}, a, y_{22})\big) = \\
&\quad \{x_2 \leftarrow a, y_{11} \leftarrow g(x_1), y_{12} \leftarrow a, y_{13} \leftarrow g(x_1), \ldots\} \qquad \Rightarrow \lambda_{2,2} = \{\} \\
\mu_{2,3} &= \mathrm{m}gu\big((g(x_1), x_2, g^2(x_1)), (y_{11}, y_{12}, g(y_{13})), (g(y_{21}), g(a), y_{22})\big) \Rightarrow \lambda_{2,3} = \{\} \\
\mu_{3,3} &= \mathrm{m}gu\big((g(x_1), x_2, g^2(x_1)), (y_{11}, y_{11}, g(y_{11})), (g(y_{21}), g(a), y_{22})\big) \\
&\quad \Rightarrow \lambda_{3,3} = \{x_1 \leftarrow a\}
\end{aligned}$$

Note that the substitution $\lambda_{2,2} = \lambda_{2,3} = \{\}$ does not have to be added to $\Lambda(3, 1)$, since the application of $\mu_{2,2}$ and $\mu_{2,3}$, respectively, to the constraint $y_{11} \neq y_{13}$ produces a trivially false disequation, i.e.: $(y_{11} \neq y_{13})\mu_{2,2} \equiv (y_{11} \neq y_{13})\mu_{2,3} \equiv g(x_1) \neq g(x_1)$. But then $\Lambda(3, 1)$ contains only the ground substitution $\lambda_{3,3} = \{x_1 \leftarrow a\}$ and, therefore, the disequation $\mathrm{D}isequ(\vartheta_{31}) \equiv x_3 \neq g^2(x_1)$ in $\mathcal{D}_3$ may be replaced by $\mathrm{D}isequ(\vartheta_{31} \circ \{x_1 \leftarrow a\}) \equiv (x_1, x_3) \neq (a, g^2(a))$. Hence, we get $\mathcal{D}_3' \equiv (z_1, z_2, z_3) = (g(x_1), x_2, x_3) \wedge (x_1, x_3) \neq (a, g^2(a))$.

Analogously to Theorem 3.1, it can be shown that the transformation of the disjuncts in a DNF according to Lemma 4.1 is actually all we need for a negation elimination procedure. Moreover, a non-deterministic version of this algorithm will allow us to derive the coNP-completeness result in Theorem 4.2

Theorem 4.1. (negation elimination from simplified formulae in DNF) *Let* $\mathcal{P} \equiv (\exists \boldsymbol{x})\big[\boldsymbol{z} = \boldsymbol{t}_1 \wedge \bigwedge_{j=1}^{l_1} \mathrm{D}isequ(\vartheta_{1j})\big] \vee \ldots \vee \big[\boldsymbol{z} = \boldsymbol{t}_n \wedge \bigwedge_{j=1}^{l_n} \mathrm{D}isequ(\vartheta_{nj})\big]$ *be an equational formula in DNF, s.t. first each disjunct has been transformed via Lemma 3.1 followed by Lemma 3.2 and then Lemma 4.1 has been applied simultaneously to all disequations. Then negation elimination from* $\mathcal{P}$ *is possible, iff* $\mathrm{V}ar(\mathrm{r}g(\vartheta_{ij})) = \emptyset$ *for every* $i \in \{1, \ldots, n\}$ *and every* $j \in \{1, \ldots, l_i\}$.

Proof. (Sketch): Exactly like in Theorem 3.1, the "if"-direction follows from the correspondence between implicit generalizations and equational formulae. The "only if"-direction is more involved. For details, see [14]. □

Theorem 4.2. (coNP-completeness) *The negation elimination problem of existentially quantified equational formulae in DNF is coNP-complete.*

Proof. (Sketch): As for the coNP-membership, let $\mathcal{P} \equiv (\exists \boldsymbol{x})\mathcal{P}_1 \vee \ldots \vee \mathcal{P}_n$ be an equational formula in DNF. W.l.o.g. we may assume that the *polynomial time* transformations from the Lemmas 3.1 and 3.2 have been applied to each disjunct $\mathcal{P}_i$. Hence, in particular, $\mathcal{P}_i$ is of the form $\big[\boldsymbol{z} = \boldsymbol{t}_i \wedge \bigwedge_{j=1}^{l_i} \mathrm{D}iseqn(\vartheta_{ij})\big]$. Then we can check via a non-deterministic version of Lemma 4.1, that negation elimination from $\mathcal{P}$ is impossible, namely: Guess a tuple $\boldsymbol{t}_i$, a substitution ϑ_{ij} and indices $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n$ and check that the resulting substitution $\lambda \in \Lambda(i,j)$ from Lemma 4.1 is non-ground.

For the coNP-hardness proof, recall that the emptiness problem of implicit generalizations is coNP-complete (cf., e.g., [7], [6], [11] or [10]), i.e.: Let $\mathcal{F}$ be a signature and let $\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n$ and $\boldsymbol{t}$ be term tuples over $\mathcal{F}$, s.t. every $\boldsymbol{s}_i$ is an instance of $\boldsymbol{t}$. Does the implicit generalization $I = \boldsymbol{t}/(\boldsymbol{s}_1 \vee \ldots \vee \boldsymbol{s}_n)$ contain no $\mathcal{F}$-ground instance? Now let an instance of the emptiness problem of implicit generalizations be given through the term tuples $\boldsymbol{s}_1 = (s_{11}, \ldots, s_{1k}), \ldots, \boldsymbol{s}_n = (s_{n1}, \ldots, s_{nk})$ and $\boldsymbol{t} = (t_1, \ldots, t_k)$, s.t. every tuple $\boldsymbol{s}_i$ is an instance of $\boldsymbol{t}$. Moreover, let $\boldsymbol{x}$ denote a vector of variables, s.t. all variables in $\boldsymbol{t}$ and in any tuple $\boldsymbol{s}_i$ are contained in $\boldsymbol{x}$. Finally let $u, v, z_1, \ldots, z_{k+2}$ be fresh, pairwise distinct variables and let $\boldsymbol{z}$ be defined as $\boldsymbol{z} = (z_1, \ldots, z_{k+2})$. Then we define the formula $\mathcal{P}$ in DNF as $\mathcal{P} \equiv (\exists \boldsymbol{x})(\exists u, v)\big[(\boldsymbol{z} = (t_1, \ldots, t_k, u, v) \wedge u \neq v) \vee \bigvee_{i=1}^{n} \boldsymbol{z} = (s_{i1}, \ldots, s_{ik}, u, u)\big]$. In [14], it is shown that the implicit generalization $I = \boldsymbol{t}/(\boldsymbol{s}_1 \vee \ldots \vee \boldsymbol{s}_n)$ is empty, iff negation elimination from $\mathcal{P}$ is possible. □

5 Equational Formulae in CNF

A straightforward negation elimination algorithm for purely existentially quantified equational formulae in CNF consists of a transformation from CNF into DNF followed by our algorithm from the previous section. Of course, in the worst case, this transformation into DNF leads to an exponential blow-up. However, by the Π_2^p-hardness shown in Theorem 5.1, we cannot expect to do much better than this anyway.

Theorem 5.1. (Π_2^p-hardness) *The negation elimination problem of existentially quantified equational formulae in CNF is Π_2^p-hard.*

Proof. (Sketch): Recall the well-known Σ_2^p-hard problem $3\mathrm{QSAT}_2$ (= quantified satisfiability with two quantifier alternations, cf. [15]), i.e.: Given sets $P = \{p_1, \ldots, p_k\}$ and $R = \{r_1, \ldots, r_l\}$ of propositional variables and a Boolean formula $E = (l_{11} \wedge l_{12} \wedge l_{13}) \vee \ldots \vee (l_{n1} \wedge l_{n2} \wedge l_{n3})$ with propositional variables in $P \cup R$, is the quantified Boolean sentence $(\exists P)(\forall R)E$ satisfiable? Now let $a \in \mathcal{F}$ denote an arbitrary constant symbol. Then we reduce such an instance of the $3\mathrm{QSAT}_2$ problem to the complementary problem of the negation elimination problem in the following way:

$\mathcal{P} \equiv (\exists \boldsymbol{x})[(d_{11} \vee d_{12} \vee d_{13} \vee z_{k+1} \neq z_{k+2}) \wedge \ldots \wedge (d_{n1} \vee d_{n2} \vee d_{n3} \vee z_{k+1} \neq z_{k+2})]$, where $\boldsymbol{z} = (z_1, \ldots, z_{k+2})$ denotes the free variables in $\mathcal{P}$, $\boldsymbol{x}$ is of the form

$\boldsymbol{x} = (x_1, \ldots, x_l)$ and the d_{ij}'s are defined as follows:

$$d_{ij} \equiv \begin{cases} z_\gamma \neq a \text{ if } l_{ij} \text{ is an unnegated propositional variable } p_\gamma \in P \\ z_\gamma = a \text{ if } l_{ij} \text{ is a negative propositional literal } \neg p_\gamma \text{ for some } p_\gamma \in P \\ x_\gamma \neq a \text{ if } l_{ij} \text{ is an unnegated propositional variable } r_\gamma \in R \\ x_\gamma = a \text{ if } l_{ij} \text{ is a negative propositional literal } \neg r_\gamma \text{ for some } r_\gamma \in R \end{cases}$$

It can be shown that negation elimination from $\mathcal{P}$ is impossible, iff $(\exists P)(\forall R)E$ is satisfiable (see [14]). □

Unfortunately, it is not clear whether the Π_2^p-membership also holds. The obvious upper bound on the negation elimination problem of equational formulae in CNF is coNEXPTIME (i.e.: transform the CNF into DNF and apply the coNP-procedure from the previous section). An exact complexity classification in case of CNF has to be left for future research.

6 Conclusion

In this paper we have presented a new negation elimination algorithm for purely existentially quantified equational formulae. The main idea of our approach was an appropriate extension of the notion of "effectively ground" disequations, which was given in [11] for the case of quantifier-free conjunctions of equations and disequations. In case of conjunctions of equations and disequations with purely existential quantifier prefix, we were thus able to decide negation elimination in polynomial time rather than by the decision procedure from [8], which has exponential complexity. For formulae in DNF our algorithm actually has exponential complexity. However, in general, it is still considerably more efficient than the algorithm from [16] for deciding whether the corresponding disjunction $\mathcal{I} = I_1 \vee \ldots \vee I_n$ of implicit generalizations has a finite explicit representation. This can be seen as follows: The algorithm from [16] has two sources of exponential complexity: One comes from a transformation rule which basically allows us to restrict the term tuples on the right-hand side of an implicit generalization I_i to the complement of another implicit generalization I_j. This transformation is very similar to our transformation in Lemma 4.1. Moreover, the algorithm from [16] contains another rule, which allows us to transform a single implicit generalization I_i in exactly the same way as the algorithm from [8]. Again, our algorithm uses the cheap transformations from the Lemmas 3.1 and 3.2 rather than the algorithm from [8]. Together with the transformations from [2] and [1] of arbitrary equational formulae into existentially quantified ones in DNF, our algorithm from Sect. 4 can be seen as a step towards a more efficient negation elimination procedure for the general case.

For existentially quantified formulae in DNF we have provided an exact complexity classification of the negation elimination problem by proving its coNP-completeness. In case of CNF, we have left a gap between the Π_2^p lower bound and the coNEXPTIME upper bound for future research.

References

1. H.Comon, C.Delor: Equational Formulae with Membership Constraints, Journal of Information and Computation, Vol 112, pp. 167-216 (1994).
2. H.Comon, P. Lescanne: Equational Problems and Disunification, Journal of Symbolic Computation, Vol 7, pp. 371-425 (1989).
3. M.Fernández: Negation Elimination in Empty or Permutative Theories, Journal of Symbolic Computation, Vol 26, pp. 97-133 (1998).
4. G.Gottlob, R.Pichler: Working with ARMs: Complexity Results on Atomic Representations of Herbrand Models, in Proceedings of LICS'99, pp. 306-315, IEEE Computer Society Press, (1999).
5. C.Kirchner, H.Kirchner, M.Rusinowitch: Deduction with symbolic constraints, Revue Française d'Intelligence Artificielle, Vol 4, pp. 9-52 (1990).
6. G.Kuper, K.McAloon, K.Palem, K.Perry: Efficient Parallel Algorithms for Anti-Unification and Relative Complement, in Proceedings of LICS'88, pp. 112-120, IEEE Computer Society Press (1988).
7. K.Kunen: Answer Sets and Negation as Failure, in Proceedings of the Fourth Int. Conf. on Logic Programming, Melbourne, pp. 219-228 (1987).
8. J.-L.Lassez, K.Marriott: Explicit Representation of Terms defined by Counter Examples, Journal of Automated Reasoning, Vol 3, pp. 301-317 (1987).
9. J.-L.Lassez, M.Maher, K.Marriott: Elimination of Negation in Term Algebras, in Proceedings of MFCS'91, LNCS 520, pp. 1-16, Springer (1991).
10. M.Maher, P.Stuckey: On Inductive Inference of Cyclic Structures, Annals of Mathematics and Artificial Intelligence Vol 15 No 2, pp. 167-208, (1995).
11. K. Marriott: Finding Explicit Representations for Subsets of the Herbrand Universe, PhD Thesis, The University of Melbourne, Australia (1988).
12. A.Martelli, U.Montanari: An efficient unification algorithm, ACM Transactions on Programming Languages and Systems, Vol 4 No 2, pp. 258-282 (1982).
13. R.Pichler: The Explicit Representability of Implicit Generalizations, to appear in Proceedings of RTA'2000, Springer (2000).
14. R.Pichler: Negation Elimination from Simple Equational Formulae, full paper, available from the author (2000).
15. L.J. Stockmeyer: The Polynomial Time Hierarchy, in Journal of Theoretical Computer Science, Vol 3, pp. 1-12 (1976).
16. M.Tajine: The negation elimination from syntactic equational formulas is decidable, in Proceedings of RTA'93, pp. 316-327, LNCS 690, Springer (1993).
17. S.Vorobyov: An Improved Lower Bound for the Elementary Theories of Trees, in Proceedings of CADE-13, LNAI 1104, pp. 275-287, Springer (1996).

Hardness of Set Cover with Intersection 1

V.S. Anil Kumar[1], Sunil Arya[2], and H.Ramesh[3]

[1] MPI für Informatik, Saarbrücken. kumar@mpi-sb.mpg.de
[2] Department of Computer Science, Hong Kong University of Science and Technology. arya@cs.ust.hk
[3] Department of Computer Science and Automation, Indian Institute of Science, Bangalore. ramesh@csa.iisc.ernet.in

Abstract. We consider a restricted version of the general Set Covering problem in which each set in the given set system intersects with any other set in at most 1 element. We show that the Set Covering problem with intersection 1 cannot be approximated within a $o(\log n)$ factor in random polynomial time unless $NP \subseteq ZTIME(n^{O(\log\log n)})$. We also observe that the main challenge in derandomizing this reduction lies in finding a hitting set for large volume combinatorial rectangles satisfying certain intersection properties. These properties are not satisfied by current methods of hitting set construction.
An example of a Set Covering problem with the intersection 1 property is the problem of covering a given set of points in two or higher dimensions using straight lines; any two straight lines intersect in at most one point. The best approximation algorithm currently known for this problem has an approximation factor of $\theta(\log n)$, and beating this bound seems hard. We observe that this problem is Max-SNP-Hard.

1 Introduction

The general Set Covering problem requires covering a given base set B of size n using the fewest number of sets from a given collection of subsets of B. This is a classical NP-Complete problem and its instances arise in numerous diverse settings. Thus approximation algorithms which run in polynomial time are of interest.

Johnson[12] showed that the greedy algorithm for Set Cover gives an $O(\log n)$ approximation factor. Much later, following advances in Probabilistically Checkable Proofs [4], Lund and Yannakakis [15] and Bellare et al. [7] showed that there exists a positive constant c such that the Set Covering problem cannot be approximated in polynomial time within a $c \log n$ factor unless $NP \subseteq DTIME(n^{O(\log\log n)})$. Feige [10] improved the approximation threshold to $(1 - o(1)) \log n$, under the same assumption. Raz and Safra[19] and Arora and Sudan [5] then obtained improved Probabilistically Checkable Proof Systems with sub-constant error probability; their work implied that the Set Covering problem cannot be approximated within a $c \log n$ approximation factor (for some constant c) unless $NP = P$.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 624–635, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Note that all the above hardness results are for general instances of the Set Covering problem and do not hold for instances when the intersection of any pair of sets in the given collection is guaranteed to be at most 1. Our motivation for considering this restriction to intersection 1 arose from the following geometric instance of the Set Covering problem.

Given a collection of points and lines in a plane, consider the problem of covering the points with as few lines as possible. Megiddo and Tamir[16] showed that this problem is NP-Hard. Hassin and Megiddo[11] showed NP-Hardness even when the lines are axis-parallel but in 3D. The best approximation factor known for this problem is $\Theta(\log n)$. Improving this factor seems to be hard, and this motivated our study of inapproximability for Set Covering with intersection 1. Note that any two lines intersect in at most 1 point.

The problem of covering points with lines was in turn motivated by the problem of covering a rectilinear polygon with holes using rectangles [13]. This problem has applications in printing integrated circuits and image compression[9]. This problem is known to be Max-SNP-Hard even when the rectangles are constrained to be axis-parallel. For this case, an $O(\sqrt{\log n})$-factor approximation algorithm was obtained recently by Anil Kumar and Ramesh[2]. However, this algorithm does not extend to the case when the rectangles need not be axis-parallel. Getting a $o(\log n)$-factor approximation algorithm for this case seems to require solving the problem of covering points with arbitrary lines, though we are not sure of the exact nature of this relationship.

Our Result. We show that there exists a constant $c > 0$ such that approximating the Set Covering problem with intersection 1 to within a factor of $c \log n$ in random polynomial time is possible only if $NP \subseteq ZTIME(n^{O(\log \log n)})$ (where $ZTIME(t)$ denotes the class of languages that have a probabilistic algorithm running in expected time t with zero error). We also give a sub-exponential derandomization which shows that approximating the Set Covering problem with intersection 1 to within a factor of $c\frac{\log n}{\log \log n}$ in deterministic polynomial time is possible only if $NP \subseteq DTIME(2^{n^{1-\epsilon}})$, for any constant $\epsilon < 1/2$.

The starting point for our result above is the Lund-Yannakakis hardness proof[15] for the general Set Covering problem. This proof uses an auxiliary set system with certain properties. We show that this auxiliary set system necessarily leads to large intersection. We then replace this auxiliary set system by another carefully chosen set system with additional properties and modify the reduction appropriately to ensure that intersection sizes stay small. The key features of the new set system are partitions of the base set into several sets of smaller size (instead of just 2 sets as in the case of the Lund-Yannakakis system or a constant number of sets as in Feige's system; small sets will lead to small intersection) and several such partitions (so that sets which "access" the same partition in the Lund-Yannakakis system and therefore have large intersection now "access" distinct partitions).

We then show how the new set system above can be constructed in randomized polynomial time and also how this randomized algorithm can be derandomized using conditional probabilities and appropriate estimators in $O(2^{n^{1-\epsilon}})$ time,

where ϵ is a positive constant, specified in Section 5. This leads to the two conditions above, namely, $NP \subseteq DTIME(2^{n^{1-\epsilon}})$ (but for a hardness of $O(\frac{\log n}{\log \log n})$) and $NP \subseteq ZTIME(n^{O(\log \log n)})$. A deterministic polynomial time construction of our new set system will lead to the quasi-NP-Hardness of approximating the Set Covering problem with intersection 1 to within a factor of $c \log n$, for some constant $c > 0$.

While the Lund-Yannakakis set system can be constructed in deterministic polynomial time using ϵ-biased limited independence sample spaces, this does not seem to be true of our set system. One of the main bottlenecks in constructing our set system in deterministic polynomial time is the task of obtaining a polynomial size hitting set for *Combinatorial Rectangles*, with the hitting set satisfying additional properties. One of these properties (the most important one) is the following: if a hitting set point has the elements i, j among its coordinates, then no other hitting set point can have both i, j among its coordinates. The only known construction of a polynomial size hitting set for combinatorial rectangles is by Linial, Luby, Saks, and Zuckerman [14] and is based on enumerating walks in a constant degree expander graph. In the full version of this paper, we show that the hitting set obtained by [14] does not satisfy the above property for reasons that seem intrinsic to the use of constant degree expander graphs.

In the full version, we also note that if the proof systems for NP obtained by Raz and Safra[19] or Arora and Sudan[5] have an additional property then the condition $NP \subseteq ZTIME(n^{O(\log \log n)})$ can be improved to $NP = ZPP$. Similarly, the statement that approximating the Set Covering problem with intersection 1 to within a factor of $c\frac{\log n}{\log \log n}$ in deterministic polynomial time is possible only if $NP \subseteq DTIME(2^{n^{1-\epsilon}})$ can be strengthened to approximation factor $c \log n$ instead of $c\frac{\log n}{\log \log n}$. The property needed of the proof systems is that the *degree*, i.e., the total number of random choices of the verifier for which a particular question is asked of a particular prover, be $O(n^{\delta})$, for some small enough constant value δ. The degree influences the number of partitions in our auxiliary proof system and therefore needs to be small. It is not clear whether existing proof systems have this property [20].

The above proof of hardness for Set Covering with intersection 1 does not apply to the problem of covering points with lines, the original problem which motivated this paper; however, it does indicate that algorithms based on set cardinalities and small pairwise intersection alone are unlikely to give a $o(\log n)$ approximation factor for this problem.

Further, our result shows that constant VC-dimension alone does not help in getting a $o(\log n)$ approximation for the Set Covering problem. This is to be contrasted with the result of Brönnimann and Goodrich[8] which shows that if the VC-dimension is a constant and an $O(\frac{1}{\epsilon})$ sized (weighted) ϵ-net can be constructed in polynomial time, then a constant factor approximation can be obtained.

The paper is organized as follows. Section 2 will give an overview of the Lund-Yannakakis reduction. Section 3 shows why the Lund-Yannakakis proof does not show hardness of Set Covering when the intersection is constrained to

be 1. Section 4 describes the reduction to Set Covering with intersection 1. This section describes a new set system we need to obtain in order to perform the reduction and shows hardness of approximation of its set cover, unless $NP \subseteq ZTIME(n^{O(\log\log n)})$. Section 5 will sketch the randomized construction of this set system. Section 6 sketches the sub-exponential time derandomization, which leads to a slightly different hardness result, unless $NP \subseteq DTIME(2^{n^{1-\epsilon}})$, $\epsilon < 1/2$. Section 7 enumerates several interesting open problems which arise from this paper.

2 Preliminaries: The Lund-Yannakakis Reduction

In this section, we sketch the version of the Lund-Yannakakis reduction described by Arora and Lund [3]. The reduction starts with a 2-Prover 1-Round proof system for Max-3SAT(5) which has inverse polylogarithmic error probability, uses $O(\log n \log\log n)$ randomness, and has $O(\log\log n)$ answer size. Here n is the size of the Max-3SAT(5) formula $\mathcal{F}$. Arora and Lund[3] abstract this proof system into the following *Label Cover* problem.

The Label Cover Problem. A bipartite graph G having $n' + n'$ vertices and edge set E is given, where $n' = n^{O(\log\log n)}$. All vertices have the same degree deg, which is polylogarithmic in n. For each edge $e \in E$, a partial function $f_e : [d] \to [d']$ is also given, where $d \geq d'$, and d, d' are polylogarithmic in n. The aim is to assign to each vertex on the left, a label in the range $1 \ldots d$, and to each vertex on the right, a label in the range $1 \ldots d'$, so as to maximize the number of edges $e = (u, v)$ satisfying $f_e(label(u)) = label(v)$. Edge $e = (u, v)$ is said to be *satisfied* by a labelling if the labelling satisfies $f_e(label(u)) = label(v)$.

The 2-Prover 1-Round proof system mentioned above ensures that either all the edges in G are satisfied by some labelling or that no labelling satisfies more than a $\frac{1}{\log^3 n}$ fraction of the edges, depending upon whether or not the Max-3SAT(5) formula $\mathcal{F}$ is satisfiable. Next, in time polynomial in the size of G, an instance $\mathcal{SC}$ of the Set Covering problem is obtained from this Label Cover problem $\mathcal{LC}$ with the following properties: if there exists a labelling satisfying all edges in G then there is a set cover of size $2n'$, and if no labelling satisfies more than a $\frac{1}{\log^3 n}$ fraction of the edges then the smallest set cover has size $\Omega(2n' \log n')$. The base set in $\mathcal{SC}$ will have size polynomial in n'. It follows that the Set Covering problem cannot be approximated to a logarithmic factor of the base set size unless $NP \subseteq DTIME(n^{O(\log\log n)})$.

Improving this condition to $NP = P$ requires using a stronger multi-prover proof system [19,5] which has a constant number of provers (more than 2), $O(\log n)$ randomness, $O(\log\log n)$ answer sizes, and inverse polylogarithmic error probability. The reduction from such a proof system to the Set Covering problem is similar to the reduction from the Label Cover to the Set Covering problem mentioned above, with a modification needed to handle more than 2 provers (this modification is described in [7]).

In this abstract, we will only describe the reduction from Label Cover to the Set Covering problem and show how we can modify this reduction to hold for the case of intersection 1. This will show that Set Covering problem with intersection 1 cannot be approximated to a logarithmic factor unless $NP \subseteq ZTIME(n^{O(\log\log n)})$. The multi-prover proof system of the previous paragraph with an additional condition can strengthen the latter condition to $NP = ZPP$; this is described in the full version.

We now briefly sketch the reduction from an instance $\mathcal{LC}$ of Label Cover to an instance $\mathcal{SC}$ of the Set Covering problem.

2.1 Label Cover to Set Cover

The following auxiliary set system given by a base set $N = \{1 \ldots n'\}$ and its partitions is needed.

The Auxiliary System of Partitions. Consider d' distinct partitions of N into two sets each, with the partitions satisfying the following property: if at most $\frac{\log n'}{2}$ sets in all are chosen from the various partitions with no two sets coming from the same partition, then the union of these sets does not cover N. Partitions with the above properties can be constructed deterministically in polynomial time [1,17]. Let P_i^1, P_i^2 respectively denote the first and second sets in the ith partition. We describe the construction of $\mathcal{SC}$ next.

Using P_i^js to construct $\mathcal{SC}$. The base set B for $\mathcal{SC}$ is defined to be $\{(e,i) | e \in E, 1 \leq i \leq n'\}$. The collection C of subsets of B contains a set $C(v,a)$, for each vertex v and each possible label a with which v can be labelled. If v is a vertex on the left, then for each a, $1 \leq a \leq d$, $C(v,a)$ is defined as $\{(e,i) | e \text{ incident on } v \wedge i \in P^1_{f_e(a)}\}$. And if v is a vertex on the right, then for each a, $1 \leq a \leq d'$, $C(v,a)$ is defined as $\{(e,i) | e \text{ incident on } v \wedge i \in P^2_a\}$.

That $\mathcal{SC}$ satisfies the required conditions can be seen from the following facts.

1. If there exists a vertex labelling which satisfies all the edges, then B can be covered by just the sets $C(v,a)$ where a is the label given to v. Thus the size of the optimum cover is $2n'$ in this case.
2. If the total number of sets in the optimum set cover is at most some suitable constant times $n' \log n'$, then at least a constant fraction of the edges $e = (u,v)$ have the property that the number of sets of the form $C(u,*)$ plus the number of sets of the form $C(v,*)$ in the optimum set cover is at most $\frac{\log n'}{2}$. Then, for each such edge e, there must exist a label a such that $C(u,a)$ and $C(v, f_e(a))$ are both in this optimum cover. It can be easily seen that choosing a label uniformly at random from these sets for each vertex implies that there exists a labelling of the vertices which satisfies an $\Omega(\frac{1}{\log^2 n'}) \geq \frac{1}{\log^3 n}$ fraction of the edges.

3 $\mathcal{SC}$ Has Large Intersection

There are two reasons why sets in the collection C in $\mathcal{SC}$ have large intersections.

Parts in the Partitions are Large. The first and obvious reason is that the sets in each partition in the auxiliary system of partitions are large and could have size $\frac{n'}{2}$; therefore, two sets in distinct partitions could have $\Omega(n')$ intersection. This could lead to sets $C(v,a)$ and $C(v,b)$ having $\Omega(n')$ common elements of the form (e,i), for some e incident on v.

Clearly, the solution to this problem is to work with an auxiliary system of partitions where each partition is a partition into not just 2 large sets, but into several small sets. The problem remains if we form only a constant number of parts, as in [10]. We choose to partition into $(n')^{1-\epsilon}$ sets, where ϵ is some non-zero constant to be fixed later. This ensures that each set in each partition has size $\theta((n')^{\epsilon}\ \text{polylog}(n))$ and that any two such sets have $O(1)$ intersection. However, smaller set size leads to other problems which we shall describe shortly.

Functions $f_e()$ are not 1-1. Suppose we work with smaller set sizes as above. Then consider the sets $C(v,a)$ and $C(v,b)$, where v is a vertex on the left and a,b are labels with the following property: for some edge e incident on v, $f_e(a) = f_e(b)$. Then each element $(e,*)$ which appears in $C(v,a)$ will also appear in $C(v,b)$, leading to an intersection size of up to $\Omega((n')^{\epsilon} * deg)$, where deg is the degree of v in G. This is a more serious problem. Our solution to this problem is to ensure that sets $C(v,a)$ and $C(v,b)$ are constructed using distinct partitions in the auxiliary system of partitions.

Next, we describe how to modify the auxiliary system of partitions and the construction of $\mathcal{SC}$ in accordance with the above.

4 $\mathcal{LC}$ to $\mathcal{SC}$ with Intersection 1

Our new auxiliary system of partitions $\mathcal{P}$ will have $d' * (deg+1) * d$ partitions, where deg is the degree of any vertex in G. Each partition has $m = (n')^{1-\epsilon}$ parts, for some $\epsilon > 0$ to be determined. These partitions are organized into d' *groups*, each containing $(deg+1) * d$ partitions. Each group is further organized into $deg+1$ *subgroups*, each containing d partitions. The first $m/2$ sets in each partition comprise its *left half* and the last $m/2$ its *right half*.

Let $P_{g,s,p}$ denote the pth partition in the sth subgroup of the gth group and let $P_{g,s,p,k}$ denote the kth set (i.e., part) in this partition. Let B_k denote the set $\cup_{g,s,p} P_{g,s,p,k}$ if $1 \le k \le m/2$, and the set $\cup_{g,s} P_{g,s,1,k}$, if $m/2 < k \le m$. We also refer to B_k as the kth *column* of $\mathcal{P}$.

We need the following properties to be satisfied by the system of partitions $\mathcal{P}$.

1. The right sides of all partitions within a subgroup are identical, i.e., $P_{g,s,p,k} = P_{g,s,1,k}$, for every $k > m/2$.
2. $P(g,s,p,k) \cap P(g',s',p',k) = \phi$ unless either $g = g', s = s', p = p'$, or, $k > m/2$ and $g = g', s = s'$. In other words, no element appears twice

within a column, modulo the fact that the right sides of partitions within a subgroup are identical.

3. $|B_k \cap B_{k'}| \leq 1$ for all k, k', $1 \leq k, k' \leq m$, $k \neq k'$.
4. Suppose N is covered using at most $\beta m \log n'$ sets in all, disallowing sets on the right sides of those partitions which are not the first in their respective subgroups. Then there must be a partition in some subgroup s such that the number of sets chosen from the left side of this partition plus the number of sets chosen from right side of the first partition in s together sum to at least $\frac{3}{4}m$.

ϵ and β are constants which will be fixed later. Let $A_{p,k} = \cup_{g,s} P_{g,s,p,k}$, for each p, k, $1 \leq p \leq d, 1 \leq k \leq m/2$. Let $D_{g,k} = \cup_s P_{g,s,1,k}$, for each g, k, $1 \leq g \leq d'$, $m/2 + 1 \leq k \leq m$. Property 2 above implies that:

5. $|A_{p,k} \cap A_{p',k}| = 0$ for all $p \neq p'$, where $1 \leq p, p' \leq d$ and $k \leq m/2$.
6. $|D_{g,k} \cap D_{g',k}| = 0$ for all $g \neq g'$, where $1 \leq g, g' \leq d'$ and $k > m/2$.

We will describe how to obtain a system of partitions $\mathcal{P}$ satisfying these properties in Section 5 and Section 6. First, we show how a set system $\mathcal{SC}$ with intersection 1 can be constructed using $\mathcal{P}$.

4.1 Using $\mathcal{P}$ to Construct $\mathcal{SC}$

The base set B for $\mathcal{SC}$ is defined to be $\{(e, i) | e \in E, 1 \leq i \leq n'\}$ as before. This set has size $(n')^2 * deg = O((n')^2 \text{ polylog}(n))$.

The collection C of subsets of B contains $m/2$ sets $C_1(v, a) \ldots C_{m/2}(v, a)$, for each vertex v on the left (in graph G) and each possible label a with which v can be labelled. In addition, it contains $m/2$ sets $C_{m/2+1}(v, a) \ldots C_m(v, a)$, for each vertex v on the right in G and each possible label a with which v can be labelled. These sets are defined as follows.

Let E_v denote the set of edges incident on v in G. We edge-colour G using $deg + 1$ colours. Let $col(e)$ be the colour given to edge e in this edge colouring. For a vertex v on the left side, and any number k between 1 and $m/2$, $C_k(v, a) = \cup_{e \in E_v} \{(e, i) | i \in P_{f_e(a), col(e), a, k}\}$. For a vertex v on the right side, and any number k between $m/2 + 1$ and m, $C_k(v, a) = \cup_{e \in E_v} \{(e, i) | i \in P_{a, col(e), 1, k}\}$.

We now give the following lemmas which state that the set system $\mathcal{SC}$ has intersection 1 and that it has a set cover of small size if and only if there exists a way to label the vertices of G satisfying several edges simultaneously. The hardness of approximation of the set cover of $\mathcal{SC}$ is given in Corollary 1, whose proof will appear in the full version.

Lemma 1. *The intersection of any two distinct sets $C_k(v, a)$ and $C_{k'}(w, b)$ is at most 1.*

Proof. Note that for $|C_k(v, a) \cap C_{k'}(w, b)|$ to exceed 1, either v, w must be identical or there must be an edge between v and w. The reason for this is that each element in $C_k(v, a)$ has the form $(e, *)$ where e is an edge incident at v while

each element in $C_{k'}(w,b)$ has the form $(e',*)$, where e' is an edge incident at w. We consider each case in turn.

Case 1. Suppose $v = w$. Then either $k \neq k'$ or $k = k', a \neq b$.

First, consider $C_k(v,a)$ and $C_{k'}(v,b)$ where $k \neq k'$ and v is a vertex in the left side. If $a = b$, observe that $C_k(v,a) \cap C_{k'}(v,a) = \phi$. So assume that $a \neq b$. The elements in the former set are of the form (e,i) where $i \in P_{f_e(a),col(e),a,k}$ and the elements of the latter set are of the form (e,j) where $j \in P_{f_e(b),col(e),b,k'}$. Note that $\cup_{e \in E_v} P_{f_e(a),col(e),a,k} \subseteq B_k$ and $\cup_{e \in E_v} P_{f_e(b),col(e),b,k'} \subseteq B_{k'}$. By Property 3 of $\mathcal{P}$, the intersection $B_k, B_{k'}$ is at most 1. However, this alone does not imply that $C_k(v,a)$ and $C_{k'}(v,b)$ have intersection at most 1, because there could be several tuples in both sets, all having identical second entries. This could happen if there are edges e_1, e_2 incident on v such that $f_{e_1}(a) = f_{e_2}(a), f_{e_1}(b) = f_{e_2}(b)$ and there had been no colouring on edges. Property 2 and the fact that $col(e_1) \neq col(e_2)$ for any two edges e_1, e_2 incident on v rule out this possibility, thus implying that $|C_k(v,a) \cap C_{k'}(v,b)| \leq 1$. The proof for the case where v is a vertex on the right is identical.

Second, consider $C_k(v,a)$ and $C_k(v,b)$, where v is a vertex on the left and $a \neq b$. Elements in the former set are of the form (e,i) where e is an edge incident on v and $i \in P_{f_e(a),col(e),a,k}$. Similarly, elements in the latter set are of the form (e,j) where $j \in P_{f_e(b),col(e),b,k}$. Note that $\cup_{e \in E_v} P_{f_e(a),col(e),a,k} \subseteq A_{a,k}$ and $\cup_{e \in E_v} P_{f_e(b),col(e),b,k} \subseteq A_{b,k}$. The claim follows from Property 5 in this case.

Third, consider $C_k(v,a)$ and $C_k(v,b)$, where v is a vertex on the right, $a \neq b$, and $k > m/2$. Elements in the former set are of the form (e,i) where e is an edge incident on v and $i \in P_{a,col(e),1,k}$. Similarly, elements in the latter set are of the form (e,j) where $j \in P_{b,col(e),1,k}$. Note that $\cup_{e \in E_v} P_{a,col(e),1,k} \subseteq D_{a,k}$ and $\cup_{e \in E_v} P_{b,col(e),1,k} \subseteq D_{b,k}$. The claim follows from Property 6 in this case.

Case 2. Finally consider sets $C_k(v,a)$ and $C_{k'}(w,b)$ where $e = (v,w)$ is an edge, v is on the left side, and w on the right. Then $C_k(v,a)$ contains elements of the form (e',i) where $i \in P_{f_{e'}(a),col(e'),a,k}$. $C_{k'}(w,b)$ contains elements of the form (e',j) where $j \in P_{b,col(e'),1,k'}$. The only possible elements in $C_k(v,a) \cap C_{k'}(w,b)$ are tuples with the first entry equal to e. Since $P_{f_e(a),col(e),a,k} \subseteq B_k$ and $P_{b,col(e),1,k'} \subseteq B_{k'}$ and $k \leq m/2, k' > m/2$, the claim follows from Properties 2 and 3 in this case. □

Lemma 2. *If there exists a way of labelling vertices of G satisfying all its edges then there exists a collection of $n'm$ sets in C which covers B.*

Proof. Let $label(v)$ denote the label given to vertex v by the above labelling. Consider the collection $C' \subset C$ comprising sets $C_1(v, label(v)) \ldots, C_{\frac{m}{2}}(v, label(v))$ for each vertex v on the left and sets $C_{\frac{m}{2}+1}(w, label(w)) \ldots, C_m(w, label(w))$ for each vertex w on the right. We show that these sets cover B. Since there are $m/2$ sets in C' per vertex, $|C'| = 2n' * \frac{m}{2} = n'm$.

Consider any edge $e = (v,w)$. It suffices to show that for every i, $1 \leq i \leq n'$, the tuple (e,i) in B is contained in either one of $C_1(v, label(v)) \ldots$, $C_{\frac{m}{2}}(v, label(v))$ or in one of $C_{\frac{m}{2}+1}(w, label(w)) \ldots, C_m(w, label(w))$. The key property we use is that $f_e(label(v)) = label(w)$.

Consider the partitions $P_{f_e(label(v)),col(e),label(v)}$ and $P_{label(w)),col(e),1}$. Since $f_e(label(v)) = label(w)$, the two partitions belong to the same group and subgroup. Since all partitions in a subgroup have the same right hand side, the element i must be present either in one of the sets $P_{label(w),col(e),label(v),k}$, where $k \leq m/2$, or in one of the sets $P_{label(w),col(e),1,k}$, where $k > m/2$. We consider each case in turn.

First, suppose $i \in P_{label(w),col(e),label(v),k}$, for some $k \leq m/2$. Then, from the definition of $C_k(v, label(v))$, $(e, i) \in C_k(v, label(v))$. Second, suppose $i \in P_{label(w),col(e),1,k}$, for some $k > m/2$. Then, from the definition of $C_k(w, label(w))$, $(e, i) \in C_k(w, label(w))$. The lemma follows. □

Lemma 3. *If the smallest collection C' of sets in C covering the base set B has size at most $\frac{\beta}{2} n' m \log n'$ then there exists a labelling of G which satisfies at least a $\frac{1}{32\beta^2 \log^2 n'}$ fraction of the edges. Recall that β was defined in Property 4 of $\mathcal{P}$.*

Proof. Given C', we need to demonstrate a labelling of G with the above property. For each vertex v, define $L(v)$ to be the collection of labels a such that $C_k(v, a) \in C'$ for some k. We think of $L(v)$ as the set of "suggested labels" for v given by C' and this will be a multiset in general. The labelling we obtain will ultimately choose a label for v from this set. It remains to show that there is a way of assigning each vertex v a label from $L(v)$ so as to satisfy sufficiently many edges.

We need some definitions. For an edge $e = (v, w)$, define $\#(e) = |L(v)| + |L(w)|$. Since the sum of the sizes of all $L(v)$s put together is at most $\frac{\beta}{2} n' m \log n'$ and since all vertices in G have identical degrees, the average value of $\#(e)$ is at most $\frac{\beta}{2} m \log n'$. Thus half the edges e have $\#(e) \leq \beta m \log n'$. We call these edges *good*.

We show how to determine a subset $L'(v)$ of $L(v)$ for each vertex v so that the following properties are satisfied. If v has a good edge incident on it then $L'(v)$ has size at most $4\beta \log n'$. Further, for each good edge $e = (v, w)$, there exists a label in $L'(v)$ and one in $L'(w)$ which together satisfy e. Clearly, random independent choices of labels from $L'(v)$ will satisfy a good edge with probability $\frac{1}{16\beta^2 \log^2 n'}$, implying a labelling which will satisfies at least a $\frac{1}{32\beta^2 \log^2 n'}$ fraction of the edges (since the total number of edges is at most twice the number of good edges), as required.

For each label $a \in L(v)$, include it in $L'(v)$ if and only if the number of sets of the form $C_*(v, a)$ in C' is at least $m/4$. Clearly, $|L'(v)| \leq \frac{\beta m \log n'}{m/4} = 4\beta \log n'$, for vertices v on which good edges are incident. It remains to show that for each good edge $e = (v, w)$, there exists a label in $L'(v)$ and one in $L'(w)$ which together satisfy e.

Consider a good edge $e = (v, w)$. Using Property 4 of $\mathcal{P}$, it follows that there exists a label $a \in L(v)$ and a label $b \in L(w)$ such that the $f_e(a) = b$ and the number of sets of the form $C_*(v, a)$ or $C_*(w, b)$ in C' is at least $3m/4$. The latter implies that the number of sets of the form $C_*(v, a)$ in C' must be at least $m/4$, and likewise for $C_*(w, b)$. Thus $a \in L'(v)$ and $b \in L'(w)$. Since $f_e(a) = b$, the claim follows. □

Corollary 1. *Set Cover with intersection 1 cannot be approximated within a factor of* $\frac{\beta \log n'}{2}$ *in random polynomial time, for some constant* β, $0 < \beta \leq \frac{1}{6}$, *unless* $NP \subseteq ZTIME(n^{O(\log \log n)})$. *Further, if the auxiliary system of partitions* $\mathcal{P}$ *can be constructed in deterministic polynomial (in* n'*) time, then approximating to within a* $\frac{\beta \log n'}{2}$ *factor is possible only if* $NP = DTIME(n^{O(\log \log n)})$.

5 Randomized Construction of the Auxiliary System $\mathcal{P}$

The obvious randomized construction is the following. Ignore the division into groups and just view $\mathcal{P}$ as a collection of subgroups. For each partition which is the first in its subgroup, throw each element i independently and uniformly at random into one of the m sets in that partition. For each partition P which is not the first in its subgroup, throw each element i which is not present in any of the sets on the right side of the first partition Q in this subgroup, into one of the first $m/2$ sets in P. Property 1 is thus satisfied directly. We need to show that Properties 2,3,4 are together satisfied with non-zero probability.

It can be shown quite easily that Property 4 holds with probability at least $1 - (\frac{1}{e})^{n'^{1-23\beta}}$, provided $\epsilon > 22\beta$. Slightly weak versions of Properties 2 and 3 (intersection bounds of 2 instead of 1) also follow immediately. This can be improved in the case of intersection 1 using the Lovasz Local Lemma, but this does not give a constant success probability and also leads to problems in derandomization. The details of these calculations appear in the full version.

To obtain a high probability of success, we need to change the randomized construction above to respect the following additional restriction (we call this Property 7): each set $P_{g,s,p,k}$ has size at most $\frac{d' * (deg+1) * dn'}{m}$, for all g, s, p, k, $1 \leq g \leq d', 1 \leq s \leq deg + 1, 1 \leq p \leq d, 1 \leq k \leq m$.

The new randomized construction proceeds as in the previous random experiment, fixing partitions in the same order as before, except that any choice of throwing an element $i \in N$ which violates Properties 2,3,7 is precluded. Property 7 enables us to show that not too many choices are precluded for each element, and therefore, this experiment stays close in behaviour to the previous one (provided $22\beta < \epsilon < 1/2$), except that Properties 2,3,7 are all automatically satisfied. The details appear in the full version.

6 Derandomization in $O(2^{n^{1-\epsilon}})$ Time

The main hurdle in derandomizing the above randomized construction in polynomial time is Property 4. There could be up to $O(2^{m \times polylog(n)}) = O(2^{(n')^{1-\epsilon'}})$ ways of choosing $\beta m \log n'$ sets from the various partitions in $\mathcal{P}$ for a constant ϵ' slightly smaller than ϵ, and we need that each of these choices fails to cover N for Property 4 to be satisfied.

For the Lund-Yannakakis system of partitions described in Section 2.1, each partition was into 2 sets and the corresponding property could be obtained deterministically using small-bias $\log n$-wise independent sample space constructions.

This is no longer true in our case. Feige's [10] system of partitions, where each partition is into several but still a constant number of parts, can be obtained deterministically using anti-universal sets [17]. However, it is not clear how to apply either Feige's modified proof system or his system of partitions to get intersection 1.

We show in the full version that enforcing Property 4 in polynomial time corresponds to constructing hitting combinatorial rectangles with certain restricted kinds of sets, though we do not know any efficient constructions for them. In this paper, we take the slower approach of using Conditional Probabilities and enforcing Property 4 by checking each of the above choices explicitly. However, note that the number of choices is superexponential in n (even though it is sub-exponential in n'). To obtain a derandomization which is sub-exponential in n, we make the following change in $\mathcal{P}$: the base set is taken to be of size n instead of n'. We use an appropriate pessimistic estimator and conditional probabilities to construct $\mathcal{P}$ with parameter n instead of n' (details will be given in the full version). This will give a gap of $\Theta(\log n)$ (instead of $\Theta(\log n')$) in the set cover instance $\mathcal{SC}$). But since the base set size in $\mathcal{SC}$ is now $O((n' * n)$ polylog$(n))$, we get a hardness of only $\Theta(\log n) = \Theta(\frac{\log n'}{\log \log n'})$ (note that the approximation factor must be with respect to the base set size) unless $NP \subset DTIME(2^{n^{1-\epsilon}})$, for any constant ϵ such that $22\beta < \epsilon < 1/2$.

7 Open Problems

A significant contribution of this paper is that it leads to several open problems.

1. Is there a polynomial time algorithm for constructing the partition system in Section 4? In the full version, we show its relation to the question of construction of hitting sets for combinatorial rectangles with certain constraints. Can a hitting set for large volume combinatorial rectangles, with the property that any two hitting set points agree in at most one coordinate, be constructed in polynomial time? Alternatively, can a different proof system be obtained, as in [10], which will require a set system with weaker hitting properties?

2. Are there instances of the problem of covering points by lines, with an integrality gap of $\Theta(\log n)$? In the full version, we show that the an integrality gap of 2 and we describe a promising construction, which might have a larger gap.

3. Are there such explicit constructions for the the Set Covering problem with intersection 1? Randomized constructions are easy for this but we do not know how to do an explicit construction.

4. Is there a polynomial time algorithm for the problem of covering points with lines which has an $o(\log n)$ approximation factor, or can super-constant hardness (or even a hardness of factor 2) be proved? In the final version, we observe that the NP-Hardness proof of Megiddo and Tamir[16] can be easily extended to a Max-SNP-Hardness proof.

References

1. N. Alon, O. Goldreich, J. Hastad, R. Perralta. Simple Constructions of Almost k-Wise Independent Random Variables. *Random Structures and Algorithms*, 3, 1992.
2. V.S. Anil Kumar and H. Ramesh. Covering Rectilinear Polygons with Axis-Parallel Rectangles. Proceedings of *31st ACM-SIAM Symposium in Theory of Computing*, 1999.
3. S. Arora, C. Lund. Hardness of Approximation. In *Approximation Algorithms for NP-Hard Problems*, Ed. D. Hochbaum, PWS Publishers, 1995, pp. 399-446.
4. S. Arora, C. Lund, R. Motwani, M. Sudan, M. Szegedy. Proof Verification and Intractability of Approximation Problems. Proceedings of *33rd IEEE Symposium on Foundations of Computer Science*, 1992, pp. 13-22.
5. S. Arora, M. Sudan. Improved Low Degree Testing and Applications. Proceedings of the *ACM Symposium on Theory of Computing*, 1997, pp. 485–495.
6. J. Beck. An Algorithmic Approach to the Lovasz Local Lemma I, *Random Structures and Algorithms*, 2, 1991, pp. 343-365.
7. M. Bellare, S. Goldwasser, C. Lund, A. Russell. Efficient Probabilistically Checkable Proofs and Applications to Approximation, Proceedings of *25th ACM Symposium on Theory of Computing*, 1993, pp. 294-303.
8. H. Brönnimann, M. Goodrich. Almost Optimal Set Covers in Finite VC-Dimension. *Discrete Comput. Geom.*, 14, 1995, pp. 263–279.
9. Y. Cheng, S.S. Iyengar and R.L. Kashyap. A New Method for Image compression using Irreducible Covers of Maximal Rectangles. *IEEE Transactions on Software Engineering*, Vol. 14, 5, 1988, pp. 651–658.
10. U. Feige. A threshold of $\ln n$ for Approximating Set Cover. *Journal of the ACM*, 45, 4, 1998, pp. 634–652.
11. R. Hassin and N. Megiddo. Approximation Algorithms for Hitting Objects with Straight Lines. *Discrete Applied Mathematics*, 30, 1991, pp. 29-42.
12. D.S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computing and Systems Sciences*, 9, 1974, pp. 256–278.
13. C. Levcopoulos. Improved Bounds for Covering General Polygons by Rectangles. Proceedings of *6th Foundations of Software Tech. and Theoretical Comp. Sc.*, LNCS 287, 1987.
14. N. Linial, M. Luby, M. Saks, D. Zuckerman. Hitting Sets for Combinatorial Rectangles. Proceedings of *25 ACM Symposium on Theory of Computing*, 1993, pp. 286–293.
15. C. Lund, M. Yannakakis. On the Hardness of Approximating Minimization Problems. Proceedings of *25th ACM Symposium on Theory of Computing*, 1993, pp. 286–293.
16. N. Megiddo and A. Tamir, On the complexity of locating linear facilities in the plane, *Oper. Res. Let*, 1, 1982, pp. 194-197.
17. M. Naor, L. Schulman, A. Srinivasan. Splitters and Near-Optimal Derandomization. Proceedings of the *36th IEEE Symposium on Foundations of Computer Science*, 1995, pp. 182–191.
18. R. Raz. A Parallel Repetition Theorem. Proceedings of the *27th ACM Symposium on Theory of Computing*, 1995, pp. 447–456.
19. R. Raz and S. Safra. A Sub-Constant Error-Probability Low-Degree test and a Sub-Constant Error-Probability PCP Characterization of NP. Proceedings of the *ACM Symposium on Theory of Computing*, 1997, pp. 475–484.
20. Madhu Sudan. Personal communication.

Strong Inapproximability of the Basic k-Spanner Problem
(Extended abstract)

Michael Elkin ⋆ and David Peleg ⋆⋆

Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel. {elkin,peleg}@wisdom.weizmann.ac.il.

Abstract. This paper studies the approximability of the sparse k-spanner problem. An $O(\log n)$-ratio approximation algorithm is known for the problem for $k = 2$. For larger values of k, the problem admits only a weaker $O(n^{1/\lfloor k \rfloor})$-approximation ratio algorithm [14]. On the negative side, it is known that the k-spanner problem is *weakly inapproximable*, namely, it is NP-hard to approximate the problem with ratio $O(\log n)$, for every $k \geq 2$ [11]. This lower bound is tight for $k = 2$ but leaves a considerable gap for small constants $k > 2$.

This paper considerably narrows the gap by presenting a *strong* (or Class III [10]) *inapproximability* result for the problem for any constant $k > 2$, namely, showing that the problem is inapproximable within a ratio of $O(2^{\log^{\epsilon} n})$, for any fixed $0 < \epsilon < 1$, unless $NP \subseteq DTIME(n^{polylog\ n})$. Hence the k-spanner problem exhibits a "jump" in its inapproximability once the required stretch is increased from $k = 2$ to $k = 2 + \delta$.

This hardness result extends into a result of $O(2^{\log^{\epsilon} n})$-inapproximability for the k-spanner problem for $k = \log^{\mu} n$ and $0 < \epsilon < 1 - \mu$, for any $0 < \mu < 1$. This result is tight, in view of the $O(2^{\log^{1-\mu} n})$-approximation ratio for the problem, implied by the algorithm of [14] for the case $k = \log^{\mu} n$. To the best of our knowledge, this is the first example for a set of Class III problems for which the upper and lower bounds "converge" in this sense.

Our main result implies also the same hardness for some other variants of the problem whose strong inapproximability was not known before, such as the uniform k-spanner problem, the unit-weight k-spanner problem, the 3-spanner augmentation problem and the "all-server" k-spanner problem for any constant k.

1 Introduction

The Sparse Spanner Problem

Graph spanners have been intensively studied in the contexts of communication networks, distributed computing, robotics and computational geometry

⋆ Supported in part by a Leo Frances Gallin Scholarship.
⋆⋆ Supported in part by grants from the Israel Ministry of Science and Art.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 636–648, 2000.
© Springer-Verlag Berlin Heidelberg 2000

[1,5,14,12,11,7,8]. Consider an unweighted n-vertex graph $G = (V, E)$. The distance between two nodes u and v in G, denoted $dist(u, v, G)$, is the minimum length of a path connecting u and v in G. A subgraph $G' = (V, E')$ of G is a *k-spanner* if $dist(u, v, G') \leq k \cdot dist(u, v, G)$ for every $u, v \in V$. We refer to k as the *stretch factor* of G'. The *sparsest k-spanner* problem is to find a k-spanner $G' = (V, E')$ with the smallest edge set E'.

The sparsest k-spanner problem is known to be NP-hard [14]. On the positive side, it is also shown in [1,2,5,14] that for every integer $k \geq 1$, every unweighted $n-$vertex graph G has a polynomial time constructible $O(k)$-spanner with at most $O(n^{1+1/k})$ edges. Hence in particular, every graph G has an $O(\log n)-$spanner with $O(n)$ edges. These results are close to the best possible in general, as implied by the lower bound given in [14].

The algorithm of [14] provides us with a *global* upper bound for sparse k-spanners, which holds for every graph. However, for specific graphs, considerably sparser spanners may exist. Furthermore, the upper bounds on sparsity given by these algorithms are small (i.e., close to n) only for large values of k. It is therefore interesting to look for approximation algorithms, that yield a near-optimal k-spanner for any given graph.

In [12], a $\log \frac{|E|}{|V|}$-approximation algorithm was presented for the unweighted 2-spanner problem. Also, since any k-spanner for an n-vertex graph requires at least $n - 1$ edges, the results of [1,2,5,14] cited above can be interpreted as providing an $O(n^{1/k})$-ratio approximation algorithm for the unweighted k-spanner problem. This implies that once the required stretch guarantee is relaxed, i.e., k is allowed to be large, the problem becomes easier to approximate. In particular, at the end of the spectrum, the unweighted k-spanner problem admits $O(1)$ approximation once the stretch requirement becomes $k = \Omega(\log n)$. Another particularly interesting intermediate point along this spectrum of k values is $k = O(\log^{\mu} n)$, $0 < \mu < 1$, where the above result implies $O(2^{\log^{1-\mu} n})$ approximation. We call this property *ratio degradation*.

On the other hand, determining the *hardness* of approximating the k-spanner problem was an open problem for quite a while. Recently, it was shown in [11] that it is NP-hard to approximate the problem by an $O(\log n)$ ratio for $k \geq 2$. This type of $\Omega(\log n)$-inapproximability is henceforth referred to as *weak inapproximability*. Hence the issue of approximability was practically resolved for $k = 2$, but a considerable gap was left for small constants $k > 2$.

On the way to resolving it, a number of different versions of the problem, harder than the original one, were formulated and proved to be hard for approximation within certain factors. Specifically, [11] introduced a certain generalization of the basic k-spanner problem called the *unit-length k-spanner* problem, and has shown that with constant stretch requirement $k \geq 5$, this generalized problem is hard to approximate with $O(2^{\log^{\epsilon} n})$ ratio. This type of $\Omega(2^{\log^{\epsilon} n})$-inapproximability is henceforth referred to as *strong inapproximability*. (In [10], problems admitting strong inapproximability results of this type are called *Class III problems*.) The latter result of [11] was later extended into hardness results for a number of other generalizations of the basic k-spanner problem [7,8]. How-

ever, the methods of [7,8] fell short of establishing the strong inapproximability of the original (unweighted) k-spanner problem, hence the exact approximability level of the original problem remained unresolved.

Main Results

In this paper we significantly narrow the gap between the known upper and lower bounds on the approximability of the unweighted k-spanner problem. This is achieved by establishing a strong inapproximability result for the basic k-spanner problem for any constant stretch requirement $k > 2$. Hence the k-spanner problem exhibits a "jump" in its inapproximability once the required stretch is increased from $k = 2$ to $k = 2+\delta$. Also our result establishes the strong hardness of the basic k-spanner problem even when restricted to bipartite graphs.

The hardness result shown in this paper extends also beyond constant k. We show that the basic k-spanner problem with stretch requirement $k = O(\log^{\mu} n)$, $0 < \mu < 1$ can not be approximated with a ratio of $O(2^{\log^{\epsilon} n})$ for any $0 < \epsilon < 1 - \mu$. Given that the k-spanner problem with $k = O(\log^{\mu} n)$, $0 < \mu < 1$ admits $O(2^{\log^{1-\mu} n})$ approximation algorithm [14], our hardness result is tight for stretch requirements $k = O(\log^{\mu} n)$, $0 < \mu < 1$.

Let us remark that this result cannot be pushed much further, since as mentioned earlier, at the end of the spectrum the problem admits an $O(1)$-approximation ratio once $k = O(\log n)$ [14], hence our result cannot be extended to $k = \log n$. It also seems very difficult (if not impossible) to strengthen our result in the sense of providing an $\Omega(n^c)$-inapproximability result for the basic spanner problem (or even to the unit-weight spanner problem), since in [8] we presented a reduction from the unit-weight spanner problem to the $MMSA_3$ *problem* (also known as the *Red-Blue problem*), and in turn, there is also a reduction from $MMSA_3$ to the $Label - Cover_{MIN}$ problem [6]. Hence an $\Omega(n^c)$-inapproximability result for some $c > 0$ for one of these problems would imply that a polynomial-time automatic prover does not exist, since its existence implies that there is a polynomial approximation within *any* polynomial factor for propositional proof length. The existence of such a polynomial time automatic prover is a long-standing open problem in proof theory (cf. [6]). Moreover, such an $\Omega(n^c)$ inapproximability result would imply a similar result for a long list of Class III problems, including MMSA, LCMIN, Min-Length-Frege-Proof, Min-Length-Resolution-Refutation, AND/OR Scheduling, Nearest-Lattice-Vector, Nearest-Codeword, Learning-Halfspaces, Quadratic-Programming, Max-π-Subgraph, Longest-Path, Diameter-Subgraph and many others (see [10,6]). In other words, it would cause classes III and IV of [10] to collapse into a single problem class.

To the best of our knowledge, the above family of $\{k - spanner\}_{k=1}^{\log^{\mu} n}$ problems, $0 < \mu < 1$, forms the first example for a Class III problem for which the upper and lower bounds converge to the same function when the parameter grows. Another family of problems, defined later on, which is also proved to enjoy a similar property, is the family of $\{MINREP_t\}_{t=1}^{\log^{\mu} n}$ problems.

As a direct consequence, we also extend the results of [8] on the strong inapproximability of a number of generalizations of the k-spanner problem beyond

what was known thus far. In particular, we establish strong inapproximability for the following problems, defined below. First, we extend the result of [8] for the *uniform* k-spanner problem with stretch requirements in the range of $1 < k \leq 3$, to any constant $k > 1$. Similarly, the strong inapproximability result for the *unit-weight* k-spanner problem with k in the range $1 < k < 3$ is extended to any constant $k > 1$. The strong inapproximability result for the k-spanner augmentation problem with k in the range $4 \leq k = O(n^\delta)$ is now extended to $3 \leq k = O(n^\delta)$. Moreover, in addition to the strong inapproximability of the *disjoint (DJ)* and *all client (AC)* (and thus *client-server (C-S)*) k-spanner problems for stretch requirements $3 \leq k = O(n^\delta)$, $0 < \delta < 1$, established in [8], we now conclude the strong inapproximability result for the *all server (AS)* k-spanner problem for any constant k.

The structure of the paper is as follows. Our main result is established by a sequence of reductions. We start from the $MAX3SAT$ problem, and use it to show the inapproximability of the $MAX3SAT_t$ problem presented in Section 2. Using this problem we establish the strong inapproximability of the $MAXREP_t$ and the $MINREP_t$ problems presented in Sections 3, 5 respectively. The last reduction transforms instances of the $MINREP_t$ problem to the $(t-1)$-spanner problem. The $MINREP_t$ problem is a restricted version of the $MINREP$ problem, shown to be strongly inapproximable in [11]. The restriction is geared at ensuring that the graphs underlying the given instances have girth greater than t, which is an essential component for facilitating the final reduction from $MINREP$ to the $(t-1)$-spanner problem.

We believe the $MINREP_t$ problem, whose strong inapproximability is established here for every constant t, may be found useful in the future for proving hardness results on other problems, that are not as hard as the (general) $MINREP$ problem.

2 The $MAX3SAT_t$ Problem

Definition 1. *For any maximization problem that attains values between* 0 *and* 1*, we say that the problem is* $(1-\delta)$-distinguishable *if there is a polynomial time algorithm that given an input I is able to distinguish between the case $\Pi(I) = 1$ and the case $\Pi(I) < 1-\delta$ (i.e., the algorithm returns 1 in the former case and 0 in the latter; there are no guarantees on the behavior of the algorithm for an intermediate input I, such that $1-\delta \leq \Pi(I) < 1$.)*

Definition 2. *For problems Π, Π' and real $\gamma > 0$, we say that $\Pi \overset{\gamma}{\propto} \Pi'$ if there exists a polynomial time reduction φ from inputs of Π to inputs of Π', for which the following two properties hold for any instance I of Π.*

(P1) *If $\Pi(I) = 1$ then $\Pi'(\varphi(I)) = 1$.*
(P2) *If $\Pi(I) < 1-\delta$ then $\Pi'(\varphi(I)) < 1-\delta/\gamma$.*

Lemma 1. *If a problem Π is $(1-\delta)$-indistinguishable unless $NP = P$ and $\Pi \overset{\gamma}{\propto} \Pi'$ then the problem Π' is $(1-\delta/\gamma)$-indistinguishable under the same assumption.*

(Proofs are omitted from this extended abstract.)

Recall that MAX3SAT is the problem of finding a truth assignment for a given boolean formula, that satisfies the maximal number of its clauses.

Lemma 2. **[10]** *$MAX3SAT$ is $(1-\delta_0)$-indistinguishable unless $NP = P$ for some $0 < \delta_0 < 1$, unless $NP = P$.*

For every instance I of the MAX3SAT problem, construct the bipartite graph $G_I = (L, R, E)$, where the set L contains a node $v(c_j)$ for every clause c_j and the set R contains a node $v(x_i)$ for every variable x_i occurring (positively or negatively) in I. An edge connects $v(c_j)$ and $v(x_i)$ if the variable x_i occurs in the clause c_j.

Define the $MAX3SAT_t$ problem to be the $MAX3SAT$ problem with an additional restriction that the girth of the graph G_I corresponding to the instance I satisfies $girth(G_I) > t$.

We start by presenting a reduction φ establishing that $MAX3SAT_t \overset{3}{\propto} MAX3SAT_{2t}$. Given an instance I of the $MAX3SAT_t$ problem, define the instance $\varphi(I)$ as follows. For every clause $c = (x_1, x_2, x_3)$, define two auxiliary variables y_1, y_2 that will be used for this clause only, and replace the clause by a set of three new clauses $s(c) = \{(x_1, \bar{y_1}),\ (y_1, x_2, \bar{y_2}),\ (y_2, x_3)\}$.

Similarly, for every clause $c = (x_1, x_2)$ we present only one auxiliary variable y_1 and replace c by a set of two new clauses $s(c) = \{(x_1, \bar{y_1}),\ (y_1, x_2)\}$. For a singleton clause c, set $s(c) = \{c\}$.

We later make use of the following two easily verified observations.

Observation 1 *(1) For every truth assignment τ for I there exists a completion τ' for $\varphi(I)$ such that for every clause c of I, c is satisfied by τ iff all the clauses in $s(c)$ are satisfied by τ'.*
(2) If the original formula I satisfies $girth(G_I) = p$ then the resulting $\varphi(I)$ satisfies $girth(G_{\varphi(I)}) \geq 2p$.

For an input I and a truth assignment τ, we denote the number of clauses in I by m_I, the number of clauses that τ satisfies in I by $m_{I,\tau}$, and the ratio between them by $\beta_{I,\tau} = m_{I,\tau}/m_I$. Let β_I^* be the ratio $\beta_{I,\tau}$ obtained under a best truth assignment τ, i.e., $\beta_I^* = \max_\tau\{\beta_{I,\tau}\}$.

We now prove that the reduction φ satisfies the properties (P1) and (P2) of Definition 2. The former follows immediately from Observation 1, implying

Lemma 3. *If $\beta_I^* = 1$ then $\beta_{\varphi(I)}^* = 1$.*

Lemma 4. *If $\beta_I^* < 1-\delta$ then $\beta_{\varphi(I)}^* < 1-\delta/3$.*

Lemma 5. $MAX3SAT_t \overset{3}{\propto} MAX3SAT_{2t}$.

As a result of this, and since $MAX3SAT$ is equivalent to $MAX3SAT_3$ (because the girth of any bipartite graph is greater than 3), we conclude, by a constant number $p = \lceil \log_2 t/3 \rceil$ of applications of the above reduction,

Lemma 6. *For any constant integer* $t > 0$, $MAX3SAT \overset{3^p}{\propto} MAX3SAT_t$.

We observe, however, that the lemma holds even if p is not a constant, but grows with n. This can be easily seen by the same proof argument.

Lemma 7. *For any constant integer* $t > 2$, *there exists a constant* $0 < \delta < 1$ *such that the* $MAX3SAT_t$ *problem is* $(1-\delta)$*-indistinguishable, unless* $NP = P$.

The above lemma can be extended to hold for non-constant values of t as well. In particular, we show the following.

Lemma 8. *There exist constants* $0 < c_1 < 1$, $c_2 > 1$ *such that for any constant* $0 < \mu < 1$ *the* $MAX3SAT_t$ *problem with* $t = \log^{c_1 \mu} n$ *is* $(1-\delta)$*-indistinguishable, unless* $NP = P$, *where* $\delta = \frac{c_2 \delta_0}{\log^{\mu} n}$ *and* δ_0 *is the constant from Lemma 2.*

3 The $MAXREP_t$ Problem

Extending the definition given in [11], the $MAXREP$ problem is defined as follows. An instance of the problem consists of the pair $\mathcal{M} = (G, \tilde{G})$, where $G(L, R, E)$ and $\tilde{G}(\tilde{L}, \tilde{R}, \tilde{E})$ are bipartite graphs. L and R are each split into a disjoint union of $\tilde{n}_l$ and $\tilde{n}_r$ sets respectively, $L = \bigcup_{i=1}^{\tilde{n}_l} U_i$ and $R = \bigcup_{i=1}^{\tilde{n}_r} W_i$. The numbers $\tilde{n}_l$ and $\tilde{n}_r$ satisfy $\tilde{n}_l \le \tilde{n}_r \le \tilde{n}_l \cdot 2^{\log^{\delta} n}$, for some $0 < \delta < 1$, where $n = |L| + |R|$. Let $N = \max\{|U_i|, |W_j| \mid 1 \le i \le \tilde{n}_l,\ 1 \le j \le \tilde{n}_r\}$.

The second component in the instance $\mathcal{M}$ is a "supergraph" $\tilde{G} = (\tilde{L}, \tilde{R}, \tilde{E})$ induced by G and the partitions of L and R, namely, with "supernodes" $\tilde{L} = \{U_1, \ldots, U_{\tilde{n}_l}\}$ and $\tilde{R} = \{W_1, \ldots, W_{\tilde{n}_r}\}$, such that two supernodes U_i and W_j are adjacent in $\tilde{G}$ iff there exists some nodes $u_i \in U_i$ and $w_j \in W_j$ which are adjacent in G, $\tilde{E} = \{(U_i, W_j) \mid \exists u_i \in U_i,\ w_j \in W_j \text{ s.t. } (u_i, w_j) \in E\}$.

A set of vertices C is said to *REP-cover* the superedge (U_i, W_j) if there exist nodes $u_1 \in C \cap U_i$ and $w_2 \in C \cap W_j$ which are adjacent in G. The set C is a $MAXREP$*-cover* for $\mathcal{M}$ if $|C \cap U_i| = 1$ for every $1 \le i \le \tilde{n}_l$ and $|C \cap W_j| = 1$ for every $1 \le j \le \tilde{n}_r$. It is required to select a $MAXREP$-cover C for $\mathcal{M}$ that REP-covers a maximal number of superedges of $\tilde{G}$.

We remark that the problem is defined here in a slightly more general form than in [11,8]. In particular, here we do not require graph-regularity on either bipartition. Also we no longer require the same number of supernodes at the left and at the right, but only that both numbers are "close." We also do not require that the same number of nodes in a supernode are used at the left and at the right.

We also define the $MAXREP_t$ problem which is the same as $MAXREP$, with the additional restriction that the instances have $girth(\tilde{G}) > t$. It is shown in [8] that the problem admits an $O(n^{\frac{2}{t+1}})$-approximation algorithm, and, in particular, that $MAXREP$ admits a $\sqrt{n}$-approximation algorithm. In the next two sections we show that the $MAXREP_t$ problem (and in fact even the more restricted $MAXREP_t^-$ problem, defined later) is $2^{\log^\epsilon n}$-indistinguishable unless $NP \not\subseteq DTIME(n^{polylog\ n})$.

Lemma 9. $MAX3SAT_t \overset{3}{\propto} MAXREP_t$.

Observe that the reduction always creates $MAXREP_t$ instances in which the supernode size is $N = O(1)$. Combining Lemmas 9, 7 and 1 we get

Lemma 10. *For any constant integer $t > 2$, there exists a constant $0 < \delta < 1$ s.t. the $MAXREP_t$ problem is $(1-\delta)$-indistinguishable, unless $NP = P$.*

Combining Lemma 9 with Lemma 8 and Lemma 1 we get

Lemma 11. *There exist constants $0 < c_1 < 1$, $c_2 > 1$ such that for any constant $0 < \mu < 1$ the $MAXREP_t$ problem with $t = \log^{c_1\mu} n$ is $(1-\delta)$-indistinguishable, unless $NP = P$, where $\delta = \frac{c_2\delta_0}{\log^\mu n}$ and δ_0 is the constant from Lemma 2.*

4 The Boosting Reduction

Definition 3. *For an instance $\mathcal{M} = (G, \tilde{G})$, $G = (L, R, E)$, $\tilde{G} = (\tilde{L}, \tilde{R}, \tilde{E})$ of $MAXREP$ and an integer $r \geq 1$, define the r-power of $\mathcal{M}$ as an instance $\mathcal{M}^r$ constructed as follows.*

1. *The set of left vertices L^r is the set of r-sequences (with repetitions) of the left vertices of G, i.e., $L^r = L \times \ldots \times L$ (r times).*
2. *Similarly, $R^r = R \times \ldots \times R$ (r times).*
3. *There is an edge between $\bar{u} = (u_1, u_2, \ldots, u_r) \in L^r$ and $\bar{w} = (w_1, w_2, \ldots, w_r) \in R^r$, if for every $i = 1, 2, \ldots, r$ there are edges (u_i, w_i) in G.*
4. *For every r-tuple of left supernodes (with repetitions) $(U_1, U_2, \ldots, U_r)$, define a supernode of G^r $\mathcal{L}(U_1, U_2, \ldots, U_r)$, that contains all the nodes $\bar{u} = (u_1, u_2, \ldots, u_r)$ such that $u_i \in U_i$ for every $i = 1, 2, \ldots, r$, and analogously for the righthand side.*

Lemma 12. *For any instance $\mathcal{M}$ of the $MAXREP_t$ problem, if $MAXREP_t(\mathcal{M}) = 1$ then $MAXREP_t(\mathcal{M}^r) = 1$.*

Lemma 13. *For every graph G and integer $r \geq 1$, $girth(G^r) \geq girth(G)$.*

The following lemma is stated implicitly in [16] for the $Label_Cover_{MAX}$ problem. (It is also stated explicitly in [10], p.419, but only for regular supergraphs.)

Lemma 14. **[16]** *Let $\mathcal{M}$ be a MAXREP-instance such that $MAXREP(\mathcal{M}) < 1-\delta$ with the size of the supernodes bounded by a constant N. Then there exists a constant $c = c(N,\delta) > 0$ such that $MAXREP(\mathcal{M}^r) \leq (MAXREP(\mathcal{M}))^{cr}$.*

It is shown in [8] that the $Label_Cover_{MAX}$ problem and the $MAXREP$ problems are equivalent, hence we conclude

Corollary 1. *Let $\mathcal{M}$ be a $MAXREP_t$-instance such that $MAXREP_t(\mathcal{M}) < 1-\delta$ with the size of the supernodes bounded by a constant N. Then there exists a constant $c = c(N,\delta) > 0$ such that $MAXREP_t(\mathcal{M}^r) \leq (MAXREP_t(\mathcal{M}))^{cr}$.*

Lemma 15. *For any instance $\mathcal{M}$ of the $MAXREP_t$ problem, if $MAXREP_t(\mathcal{M}) < 1-\delta$ then $MAXREP_t(\mathcal{M}^r) < (1-\delta)^{O(r)}$.*

Lemma 16. *For any $0 < \epsilon < 1$, the problem $MAXREP_t$ is $\frac{1}{2^{\log^\epsilon n}}$-indistinguishable (or $(1-\delta)$-indistinguishable with $\delta = 1 - \frac{1}{2^{\log^\epsilon n}}$), unless $NP \subseteq DTIME$ $(n^{polylog\ n})$.*

Lemma 17. *There exists a constant $0 < c_1 < 1$ such that for any constant $0 < \mu < 1$ and for any constant $0 < \epsilon < 1-\mu$ the $MAXREP_t$ problem with $t = \log^{c_1\mu} n$ is $2^{-log^\epsilon n}$-indistinguishable (or $(1-\delta)$-indistinguishable, where $\delta = 1 - 2^{-log^\epsilon n}$), unless $NP \subseteq DTIME(n^{polylog\ n})$.*

5 The $MINREP_t$ Problem

The $MINREP$ problem is the minimization version of the $MAXREP$ defined in Section 3. Both problems were introduced in [11], and the $MINREP$ problem was studied in [8]. In the $MINREP$ problem, any number of nodes can be taken into REP-cover from one supernode (in contrast to the $MAXREP$ problem, where we insist on taking at most one representative from each supernode), but it is required that the REP-cover should be *proper*, i.e., it should cover *all* the superedges. The problem is to find a minimal size proper REP-cover. We consider also the $MINREP_t$ problem, which is the $MINREP$ problem restricted to instances whose supergraph $\tilde{G}$ has $girth(\tilde{G}) > t$.

The $MINREP$ problem is proved to be strongly inapproximable in [11]. In [8] we have shown that this problem admits a $\sqrt{n}$-approximation algorithm, and more generally, that the $MINREP_t$ problem admits an $n^{2/(t+1)}$-approximation ratio.

In this section we show that the $MINREP_t$ problem is strongly inapproximable, unless $NP \subseteq DTIME(n^{polylog\ n})$. This is done by presenting a reduction from the $MAXREP_t$ problem, discussed in the previous sections, to the $MINREP_t$ problem, and then using Lemma 16.

We make use of the following technical lemma.

Lemma 18. *Let* $a_1, \ldots, a_{\tilde{n}_l}, b_1, \ldots, b_{\tilde{n}_r} \geq 0$ *such that* $\sum_{i=1}^{\tilde{n}_l} a_i + \sum_{j=1}^{\tilde{n}_r} b_j = z$. *Then* $\sum_{i=1}^{\tilde{n}_l} \sum_{j=1}^{\tilde{n}_r} \frac{1}{a_i b_j} \geq \frac{4\tilde{n}_l^2 \tilde{n}_r^2}{z^2}$.

Lemma 19. *Given an approximation algorithm* A *for the* $MINREP_t$ *problem with approximation ratio* ρ*, for any* $0 < \epsilon < 1$*, the following properties hold for any instance* $\mathcal{M}$ *of the* $MAXREP_t$ *problem.*

(Q1) *If* $MAXREP_t(\mathcal{M}) = 1$ *then* $A(\mathcal{M}) < \rho(\tilde{n}_l + \tilde{n}_r)$.
(Q2) *If* $MAXREP_t(\mathcal{M}) < 2^{-\log^\epsilon n}$ *then* $A(\mathcal{M}) > 2 \cdot 2^{\frac{1}{2}\log^\epsilon n} \cdot \tilde{n}_l \tilde{n}_r$.

We observe that the randomized argument used in the proof can be derandomized using the method of conditional probabilities.

Theorem 2. *For any* $0 < \epsilon < 1$ *there is no approximation algorithm for the* $MINREP_t$ *problem with approximation ratio* $\rho < 2^{\log^\epsilon n}$*, unless* $NP \subseteq DTIME(n^{polylog\ n})$.

Lemma 20. *There exists a constant* $0 < c_1 < 1$ *such that for any constant* $0 < \mu < 1$ *and for every constant* $0 < \epsilon < 1 - \mu$*, there is no algorithm for the* $MINREP_t$ *problem with* $t = \log^{c_1 \mu} n$ *with an approximation ratio* $\rho < 2^{\log^\epsilon n}$*, unless* $NP \subseteq DTIME(n^{polylog\ n})$.

The proof is the same as for Theorem 2, except that at the end we use Lemma 17.

6 The k-Spanner Problem

The hardness of the basic k-spanner problem for $k \geq 3$ is proven by an extension of the reduction from the $MINREP$ problem to DJ C-S k-spanner problem from [8]. Let $\mathcal{M} = (G, \tilde{G})$ be a $MINREP_{k+1}$ instance, where $G = (L, R, E)$, $\tilde{G} = (\tilde{L}, \tilde{R}, \tilde{E})$, $\tilde{L}$ is a collection of disjoint subsets of L and $\tilde{R}$ is a collection of disjoint subsets of R. It can be checked in a polynomial time whether there exists a REP-cover C for $\mathcal{M}$, by checking whether $L \cup R$ REP-covers all the superedges. If $L \cup R$ does not REP-cover all the superedges, the instance has no MINREP-cover. Thus, without loss of generality, we assume that $L \cup R$ does REP-cover all the superedges and, in particular, there exists a REP-cover for G. We set $\tilde{n} = \tilde{n}_l + \tilde{n}_r$ and $k_l = \lfloor \frac{k-3}{2} \rfloor$, $k_r = \lceil \frac{k-3}{2} \rceil$, and $x = n^2/\tilde{n}$, and build the graph $\bar{G}$ as follows (see Fig. 1). For any integer $z > 0$ we denote $[z] = \{1, \ldots, z\}$. Define new vertex sets

$$S = \{s_{ii'}^p \mid i \in [\tilde{n}_l], i' \in [k_l], p \in [x]\} \text{ and } T = \{t_{jj'}^p j \in [\tilde{n}_r], j' \in [k_r], p \in [x]\} ,$$

and set $\bar{V} = L \cup R \cup S \cup T$ and $\bar{E} = E \cup E_{sU} \cup E_{tW} \cup E_{\tilde{G}} \cup E_M$, where

$$E_M = \{(s_{ii'}^p, s_{i(i'+1)}^p) \mid p \in [x], i \in [\tilde{n}_l], i' \in [k_l - 1]\}$$

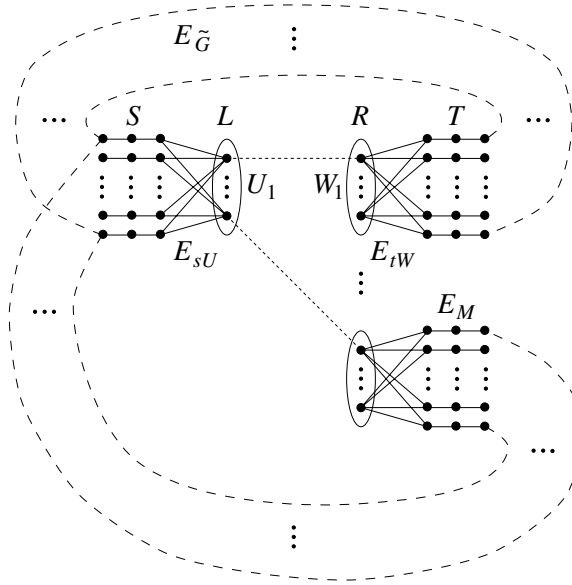


Fig. 1. The graph G.

$$\cup \ \{(t^p_{jj'}, t^p_{j(j'+1)}) \mid p \in [x], j' \in [k_r - 1], j \in [\tilde{n}_r]\} \ ,$$

$$E_{sU} = \{(s^p_{i1}, u_i) \mid u_i \in U_i, i \in [\tilde{n}_l], p \in [x]\}$$

$$E_{tW} = \{(w_j, t^p_{j1}) \mid w_j \in W_j, j \in [\tilde{n}_r], p \in [x]\},$$

$$E_{\tilde{G}} = \bigcup_{p=1}^{x} E^p_{\tilde{G}} \ , \quad \text{where}$$

$$E^p_{\tilde{G}} = \{(s^p_{ik_l}, t^p_{jk_r}) \mid (U_i, W_j) \in \tilde{G}\} \ .$$

Denote also $E^{i,j}_{\tilde{G}} = \{(s^p_{ik_l}, t^p_{jk_r}) \mid (U_i, W_j) \in \tilde{G}, 1 \le p \le x\}$. Observe that for $k = 3$, $k_l = k_r = 0$ and thus $E_M = \emptyset$.

Note also that each set of edges $E^p_{\tilde{G}}$ is an isomorphic copy of the supergraph $\tilde{G}$. The graph is built in such a way that all the edges except $E_{\tilde{G}}$ can be easily spanned, and the proof is based on establishing a connection between the size of the REP-cover and the number of edges required for spanning the edges of $E_{\tilde{G}}$. It is easy to see that for even values of k all the cycles in $\bar{G}$ are of even length. Thus in this case $\bar{G}$ is a bipartite graph. This enables us to show our hardness result for the basic k-spanner problem, even when restricted to bipartite graphs. The following two observations are immediate.

Observation 3 *(1) No path k-spanning an $E_{\tilde{G}}$ edge can pass through some other $E_{\tilde{G}}$ edge and an E edge.*
(2) No path using only $E_{\tilde{G}}$ edges can k-span another $E_{\tilde{G}}$ edge.

The first observation holds because otherwise the length of the path would be longer than k. The second follows because $girth(\tilde{G}) > k+1$.

The next lemma claims, intuitively, that the copies $E^p_{\tilde{G}}$ cannot help each other in spanning their edges.

Lemma 21. *No path P such that $P \cap E^p_{\tilde{G}} \neq \emptyset$ can be used to k-span an edge from $E^q_{\tilde{G}}$, for $p \neq q$.*

Corollary 2. *No path P such that $P \cap E^{i,j}_{\tilde{G}} \neq \emptyset$ can k-span any $E^{i,j}_{\tilde{G}}$ edge e such that $e \notin P$.*

Lemma 22. *Let P be a simple path such that $P \cap E_{\tilde{G}} \neq \emptyset$. Let the edge e satisfy $e \in E_{\tilde{G}} \setminus P$. Then P does not k-span the edge e.*

It follows that there are only two ways to k-span the $E_{\tilde{G}}$ edges in $\bar{G}$, namely, either to self-span them, or to span them by a *direct* path via the original edgeset E, namely, a path of the type
$(s^p_{ik_l}, s^p_{i(k_l-1)}, \ldots, s^p_{i1}, u^m_i, w^{m'}_j, t^p_{j1}, t^p_{j2}, \ldots, t^p_{jk_r})$.
Note that the length of such a path is exactly $k_l + k_r + 3 = k$. Intuitively, edges in E are very cheap, and the really crucial edges are the stars from u_i's and w_j's to the first layer of S and T. Thus, we would really want to minimize the number of such stars, which would correspond to the needed number of nodes taken in the REP-cover.

A k-spanner H for the graph $\bar{G}$ is called a *proper* spanner if it does not use any edge of $E_{\tilde{G}}$, i.e., $H \cap E_{\tilde{G}} = \emptyset$.

Corollary 3. *A proper k-spanner H k-spans all the edges of $E_{\tilde{G}}$ by direct paths.*

A main observation made next is that forbidding the use of $E_{\tilde{G}}$ edges does not degrade the spanner quality by much.

Lemma 23. *Any k-spanner H for $\bar{G}$ can be converted in polynomial time to a proper k-spanner H' of size $|H'| \leq 6|H|$.*

Lemma 24. *Given a k-spanner H for $\bar{G}$ there is a polynomial time constructible REP-cover C for $\mathcal{M} = (G, \tilde{G})$ of size $|C| \leq \frac{6|H|}{x}$.*

For the opposite direction we prove the following lemma.

Lemma 25. *Given a REP-cover C for the $MINREP$ instance $\mathcal{M} = (G, \tilde{G})$, there is a poly-time constructible k-spanner H of $\bar{G}$ of size $|H| \leq (k-1)x|C|$.*

Now we are ready to prove our main result.

Theorem 4. *For every constant $k > 2$, the basic k-spanner problem is strongly inapproximable, even when restricted to bipartite graphs, unless $NP \subseteq DTIME(n^{polylog\ n})$.*

Again, we also generalize our result to the k-spanner problem with a stretch requirement $k = \log^{O(\mu)} n$.

Theorem 5. *There exists a constant $0 < c_1 < 1$ such that for any constant $0 < \mu < 1$ and for every constant $0 < \epsilon < 1 - \mu$, the basic $(\log^{c_1 \mu} n)$-spanner problem is $2^{-\log^{\epsilon} n}$-inapproximable, unless $NP \subseteq DTIME(n^{polylog\ n})$.*

The proof of the theorem is analogous to the proof of Theorem 4 with only slight changes in the analysis. It uses Lemma 20 except of 2 at the end.

Improving the exponent
We note that the lower bound of Theorem 5 is tight in the sense that the $O(n^{1/k})$-approximation algorithm of [14] supplies an approximation ratio of $n^{\frac{1}{c_1 \log^{\mu} n}} = 2^{\frac{1}{c_1} \log^{1-\mu} n}$ for the k-spanner problem with $k = \log^{c_1 \mu} n$. Furthemore, it cannot be extended to $k = \log n$, because the problem becomes $O(1)$-approximable for this stretch requirement [14]. However, the lower bound as is applies only to the k-spanner problem with the stretch requirement $\log^{\mu'} n$ with $0 < \mu' < c_1$, and in the above analysis $c_1 = \log_3 2$. Intuitively, this value of c_1 follows from the fact that the reduction in Section 2 "loses" a factor of 3 on each iteration.

Let us point out, however, that we are able to show that after sufficiently many iterations the reduction loses only a factor of $2+\eta$ on each iteration, where $\eta > 0$ is arbitrarily small. This observation can be used in order to extend the result for any constant $0 < c_1 < 1$. The exact analysis will appear in the full paper.

Acknowledgements

The authors would like to thank Uri Feige and Ran Raz for helpful discussions and Oleg Hasanov for supplying a short and elegant proof for Lemma 18.

References

1. I. Althöfer, G. Das, D. Dobkin, D. Joseph and J. Soares", On Sparse Spanners of Weighted Graphs, *Discrete & Computational Geometry*, **9**, (1993), pp. 81–100
2. Baruch Awerbuch, Alan Baratz, and David Peleg. Efficient broadcast and light-weight spanners. Technical Report CS92-22, Weizmann Institute, October 1992.
3. B. Bollobas, *Extremal Graph Theory.* Academic Press, 1978.
4. Robert Carr, Srinivas Doddi, Goran Konjevod, and Madhav Marathe. On the red-blue set cover problem. In *Proc. 11th ACM-SIAM Symp. on Discrete Algorithms*, 2000.
5. B. Chandra, G. Das, G. Narasimhan and J. Soares, New Sparseness Results on Graph Spanners, *Proc 8th ACM Symp. on Computat. Geometry*, 192-201, 1992.
6. I. Dinur and S. Safra, On the hardness of Approximating Label Cover, *Electronic Colloquium on Computational Complexity*, Report No. 15 (1999)
7. Y. Dodis and S. Khanna, Designing Networks with Bounded Pairwise Distance, *Proc. 30th ACM Ann. Symp. of Theory of Computing*, 1999.

8. M.-L. Elkin and D. Peleg, The Hardness of Approximating Spanner Problems, *Proc. 17th Symp. on Theoretical Aspects of Computer Science*, Lille, France, 2000.
9. M.-L. Elkin and D. Peleg, The Client-Server 2-Spanner Problem and Applications to Network Design, Technical Report MCS99-24, the Weizmann Institute, 1999.
10. D. Hochbaum *Approximation Algorithms for NP-hard Problems*, PWS Publishing Company, Boston, 1997.
11. G. Kortsarz, On the Hardness of Approximating Spanners, *Proc. APPROX.*, LNCS, Vol. 1444, pp. 135-146, Springer-Verlag, New York/Berlin, 1998.
12. G. Kortsarz and D. Peleg, Generating Sparse 2-Spanners. *J. Algorithms*, **17** (1994) 222-236.
13. D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM, Philadelphia, PA, 2000, in press.
14. D. Peleg and A. Schäffer, Graph Spanners, *J. Graph Theory* **13** (1989), 99-116.
15. C. Papadimitriou and M. Yannakakis. Optimization, Approximation and Complexity Classes, *J. of Computer and System Sciences*, **43**, 1991, pp. 425-440.
16. R. Raz, A parallel repetition theorem. *Proc. 27th ACM STOC* pp.447-456, 1995.

Infinite Series-Parallel Posets: Logic and Languages

Dietrich Kuske

Institut für Algebra, Technische Universität Dresden,
D-01062 Dresden, Germany
kuske@math.tu-dresden.de
Fax +49 351 463 4235

Abstract. We show that a set of uniformly width-bounded infinite series-parallel pomsets is ω-series-rational iff it is axiomatizable in monadic second order logic iff it is ω-recognizable. This extends recent work by Lodaya and Weil on sets of finite series-parallel pomsets in two aspects: It relates their notion of series-rationality to logical concepts, and it generalizes the equivalence of recognizability and series-rationality to infinite series-parallel pomsets.

1 Introduction

In theoretical computer science, finite words are a classical concept that is used to model the behavior of a sequential system. In this setting, the atomic actions of the system are considered as letters of an alphabet Γ. A natural operation on such sequential behaviors is the concatenation; it models that, after finishing one task, a system can start another one. Therefore, the natural mathematical model is that of a (free) monoid $\Gamma^\star$. To model not only the behavior of a sequential system, but also allow parallelism, labeled partially ordered sets or pomsets were suggested [11,18,10]. In this setting, there is not only one, but there are (at least) two natural operations: A parallel system can start a new job after finishing the first one, or it can perform two jobs in parallel. These two operations are mathematically modeled by the sequential and the parallel product on pomsets: In the sequential product, the second pomset is set on top of the first. Complementary, in the parallel product, the two pomsets are put side by side. Thus, in the sequential product all events of the first factor are related to all events of the second while in the parallel product no additional relations are inserted. Another approach is that of Mazurkiewicz traces. Here, the sequentiality/parallelism is dictated by a fixed dependence relation on the set of actions. Therefore, the trace product (w.r.t. a given dependence relation) of two pomsets relates only dependent events of the two factors.

Pomsets that one obtains by the sequential and the parallel product from the singletons are known as series-parallel pomsets. It was shown that finite series-parallel pomsets are precisely those pomsets that do not contain a subposet of the form N (hence their alternative name "N-free posets") [10]. Together with the

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 648–662, 2000.
© Springer-Verlag Berlin Heidelberg 2000

sequential and the parallel product, the finite N-free pomsets form an algebra, called sp-algebra, that generalizes the free monoid $\Gamma^\star$. The equational theory of this algebra was considered in [10]. Pomsets constructed from the singletons by the trace product are called traces. Together with the trace product, they form a monoid, called trace monoid. See [4] for a recent survey on the many results known on traces.

Several models of computational devices are known in theoretical computer science. The probably simplest one is that of a finite automaton, i.e. a finite state device capable of accepting or rejecting words. Several characterizations of the accepting power of finite automata are known: A set of words L can be accepted by a finite automaton if it is rational (Kleene), axiomatizable in monadic second order logic (Büchi) or recognizable by a homomorphism into a finite monoid (Myhill-Nerode). Several attempts have been made to resume the success story of finite automata to pomsets, i.e. to transfer the nice results from the setting of a sequential machine to concurrent systems. For traces, this was achieved to a large extend by asynchronous (cellular) automata [22] (see [5] for an extension to pomsets without autoconcurrency). For N-free pomsets, Lodaya and Weil introduced branching automata. In [15,16] they were able to show that a set of finite width-bounded N-free pomsets is rational iff series-rational (i.e. can be constructed from the singletons by union, sequential and parallel product and by the sequential iteration) iff recognizable (i.e. saturated by a homomorphism into a finite sp-algebra). This was further extended in [14] by the consideration of sets that are not uniformly width-bounded.

While finite words are useful to deal with the behavior of terminating systems, ω-words serve as a model for the behavior of nonterminating systems. Most of the results on recognizable languages of finite words were extended to ω-words (see [21] for an overview). For traces, this generalization was fruitful, too [9,6,3]. Bloom and Ésik [1] considered the set of pomsets obtained from the singletons by the sequential and the parallel product and by the sequential ω-power. In addition, Ésik and Okawa [7] allowed the parallel ω-power. They obtained inner characterizations of the pomsets obtained this way and considered the equational theory of the corresponding algebras.

This paper deals with the set of pomsets that can be obtained by the sequential and the parallel product as well as by the infinite sequential product of pomsets. First, we show a simple characterization of these pomsets (Lemma 1). The main part of the paper is devoted to the question whether Büchi's correspondence between monadic second order logic on ω-words and recognizable sets can be transfered to the setting of (infinite) N-free pomsets. Our main result, Theorem 16, states that this is indeed possible. More precisely, we consider ω-series-rational sets, i.e. sets that can be constructed from finite sets of finite N-free pomsets by the operation of sequential and parallel concatenation, sequential iteration, sequential ω-iteration and union (without the ω-iteration, this class was considered in [15,16]). We can show that a set of infinite N-free pomsets is ω-series-rational if and only if it can be axiomatized in monadic second order logic and is width-bounded. Our proof relies on a suitable (algebraic)

definition of recognizable sets of infinite N-free pomsets and on a deep result from the theory of infinite traces [6].

Recall that Courcelle [2] considered the counting monadic second order logic on graphs of finite tree width. In this setting, a set of finite graphs is axiomatizable in Courcelle's logic if and only if it is "recognizable" [13]. It is not difficult to show that any ω-series-rational set of N-free pomsets is axiomatizable in this logic. If one tried to prove the inverse implication, i.e. started from an axiomatizable set of N-free pomsets, one would yield a rational set of terms over the parallel and the sequential product. But, as usual in term languages, this set makes use of an extended alphabet. Therefore, it is not clear how to construct a series-rational expression without additional variables from this rational term language. For this difficulty, we chose to prove our main result using traces and not Courcelle's approach.

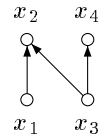
Let us finish this introduction with some open problems that call for an investigation: First, we obtained only a relation between algebraically recognizable, monadically axiomatizable, and ω-series-rational sets. It would be interesting to have a characterization in terms of branching automata, too. To this purpose, one first has to extend them in such a way that branching automata can run on infinite N-free pomsets. Second, we would have liked to incorporate the parallel iteration or even the parallel ω-power in the construction of rational sets. This easily allows the construction of sets that cannot be axiomatized in monadic second order logic. Therefore, one could try to extend the expressive power of this logic suitably.

2 Basic Definitions

2.1 Order Theory

Let $(V, \leq)$ be a partially ordered set. We write $x \parallel y$ for elements $x, y \in V$ if they are incomparable. A set $A \subseteq V$ is an *antichain* provided the elements of A are mutually incomparable. The *width* of the partially ordered set $(V, \leq)$ is the least cardinal $w(V, \leq)$ such that $|A| \leq w(V, \leq)$ for any antichain A. If $w(V, \leq)$ is a natural number, we say $(V, \leq)$ *has finite width*. Note that there exist partially ordered sets that contain only finite antichains, but have infinite width. We write $x \lessdot y$ for $x, y \in V$ if $x < y$ and there is no element properly between x and y. Furthermore, $\downarrow y$ denotes the principal ideal $\{x \in V \mid x \leq y\}$ generated by $y \in V$.

An *N-free poset* $(V, \leq)$ is a nonempty, at most countably infinite partially ordered set such that the partially ordered set $(N, \leq_N)$ cannot be embedded into $(V, \leq)$ (cf. picture below), any antichain in $(V, \leq)$ is finite, and $\downarrow x$ is finite for any $x \in V$. Let Γ be an alphabet, i.e. a nonempty finite set. Then $\mathrm{NF}^\infty(\Gamma)$ denotes the set of all Γ-labeled N-free posets $(V, \leq, \lambda)$. These labeled posets are called *N-free pomsets*. Let $\mathrm{NF}(\Gamma)$ denote the set of finite N-free pomsets over Γ.



The poset $(N, \leq_N)$

Next, we define the sequential and the parallel product of Γ-labeled posets: Let $t_1 = (V_1, \leq_1, \lambda_1)$ and $t_2 = (V_2, \leq_2, \lambda_2)$ be Γ-labeled posets with $V_1 \cap V_2 = \emptyset$. The *sequential product* $t_1 \cdot t_2$ of t_1 and t_2 is the Γ-labeled partial order

$$(V_1 \cup V_2, \leq_1 \cup \leq_2 \cup V_1 \times V_2, \lambda_1 \cup \lambda_2).$$

Thus, in $t_1 \cdot t_2$, the labeled poset t_2 is put on top of the labeled poset t_1. On the contrary, the *parallel product* $t_1 \parallel t_2$ is defined to be

$$(V_1 \cup V_2, \leq_1 \cup \leq_2, \lambda_1 \cup \lambda_2),$$

i.e. here the two partial orders are set side by side. By $\mathrm{SP}(\Gamma)$, we denote the least class of Γ-labeled posets containing the singletons that is closed under the application of the sequential product $\cdot$ and the parallel product $\parallel$.

To construct infinite labeled posets, we extend the sequential product $\cdot$ naturally to an infinite one as follows: For $i \in \omega$, let $t_i = (V_i, \leq_i, \lambda_i)$ be mutually disjoint Γ-labeled posets. Then the infinite sequential product is defined by

$$\prod_{i \in \omega} t_i = (\bigcup_{i \in \omega} V_i, \bigcup_{i \in \omega} \leq_i \cup \bigcup_{\substack{i,j \in \omega \\ i<j}} V_i \times V_j, \bigcup_{i \in \omega} \lambda_i).$$

By $\mathrm{SP}^\infty(\Gamma)$, we denote the least class $\mathcal{C}$ of Γ-labeled posets such that

- $\mathrm{SP}(\Gamma) \subseteq \mathcal{C}$,
- $s, t \in \mathcal{C}$ implies $s \parallel t \in \mathcal{C}$,
- $s, t \in \mathcal{C}$ and s finite imply $s \cdot t \in \mathcal{C}$, and
- $t_i \in \mathcal{C}$ finite for $i \in \omega$ implies $\prod_{i \in \omega} t_i \in \mathcal{C}$.

Thus, a Γ-labeled poset belongs to $\mathrm{SP}^\infty(\Gamma)$ if it can be constructed from the finite Γ-labeled pomsets applying the sequential product, the parallel product or the infinite product.

Based on results from [1], we extend the known equality $\mathrm{SP}(\Gamma) = \mathrm{NF}(\Gamma)$ [10] to infinite Γ-labeled posets:

Lemma 1. *Let Γ be an alphabet. Then* $\mathrm{SP}^\infty(\Gamma) = \mathrm{NF}^\infty(\Gamma)$.

Proof. By induction on the construction of an element of $\mathrm{SP}^\infty(\Gamma)$, one shows the inclusion $\mathrm{SP}^\infty(\Gamma) \subseteq \mathrm{NF}^\infty(\Gamma)$. For the converse inclusion, let $t \in \mathrm{NF}^\infty(\Gamma)$. We may assume that t is connected. By [1, Lemma 4.10], either t is an infinite sequential product of finite N-free pomsets, or there exist $s \in \mathrm{NF}(\Gamma)$ and $t_1, t_2 \in \mathrm{NF}^\infty(\Gamma)$ with $t = s \cdot (t_1 \parallel t_2)$. Then we can proceed inductively with t_1 and t_2. This inductive decomposition will eventually terminate since antichains in t are finite. Since $\mathrm{NF}(\Gamma) = \mathrm{SP}(\Gamma)$, this finishes the proof. $\square$

The sequential, the parallel and the infinite sequential products can easily be extended to sets of (finite) N-free pomsets as follows: Let $S \subseteq \mathrm{NF}(\Gamma)$ and $S', T' \subseteq \mathrm{NF}^\infty(\Gamma)$. Then we define

$$S \cdot T' := \{s \cdot t \mid s \in S, t \in T'\}, \qquad S^+ := \{s_1 \cdot s_2 \cdots s_n \mid n > 0, s_i \in S\},$$
$$S' \parallel T' := \{s \parallel t \mid s \in S, t \in T\} \text{ and } S^\omega := \{\textstyle\prod_{i\in\omega} s_i \mid s_i \in S\}.$$

The class of *series-rational languages* [15,16] is the least class $\mathcal{C}$ of subsets of $\mathrm{NF}(\Gamma)$ such that

- $\{s\} \in \mathcal{C}$ for $s \in \mathrm{NF}(\Gamma)$, and
- $S \cup T, S \cdot T, S \parallel T, S^+ \in \mathcal{C}$ for $S, T \in \mathcal{C}$.

Note that we do not allow the iteration of the parallel product in the construction of series-rational languages. Therefore, for any series-rational language S there exists an $n \in \omega$ with $w(s) \leq n$ for any $s \in S$, i.e. any series-rational language is *width-bounded*.

The class of *ω-series-rational languages* is the least class $\mathcal{C}$ of subsets of $\mathrm{NF}^\infty(\Gamma)$ such that

- $\{s\} \in \mathcal{C}$ for $s \in \mathrm{NF}(\Gamma)$,
- $S \cup T, S \parallel T \in \mathcal{C}$ for $S, T \in \mathcal{C}$, and
- $S^+, S^\omega, S \cdot T \in \mathcal{C}$ for $S, T \in \mathcal{C}$ and $S \subseteq \mathrm{NF}(\Gamma)$.

For the same reason as for series-rational languages, any ω-series-rational language is width-bounded. It is easily seen that the series-rational languages are precisely those ω-series-rational languages that contain only finite labeled posets.

2.2 Traces

We recall the basic definitions and some results from the theory of Mazurkiewicz traces since they will be used in our proofs, in particular in Section 4.2.

A *dependence alphabet* (Γ, D) is a finite set Γ together with a binary reflexive and symmetric relation D that is called *dependence relation*. The complementary relation $I = \Gamma^2 \setminus D$ is the *independence relation*. From a dependence alphabet, we define a binary operation $*$ on the set of Γ-labeled posets as follows: Let $t_i = (V_i, \leq_i, \lambda_i)$ be disjoint Γ-labeled posets ($i = 1, 2$). Furthermore, let $E = \{(x, y) \in V_1 \times V_2 \mid (\lambda_1(x), \lambda_2(y)) \in D\}$. Then

$$t_1 * t_2 := (V_1 \cup V_2, \leq_1 \cup (\leq_1 \circ E \circ \leq_2) \cup \leq_2, \lambda_1 \cup \lambda_2)$$

is the *trace product* of t_1 and t_2 relative to (Γ, D). Let $\mathbb{M}(\Gamma, D)$ denote the least class of Γ-labeled posets closed under the application of the trace product $*$ that contains the singletons. The elements of $\mathbb{M}(\Gamma, D)$ are called *traces*. The set $\mathbb{M}(\Gamma, D)$ together with the trace product as a binary operation is a semigroup (it is no monoid since we excluded the empty poset from our considerations). Note that $\mathbb{M}(\Gamma, D)$ consists of finite posets, only. One can show that a finite

nonempty Γ-labeled poset $(V, \leq, \lambda)$ belongs to $\mathbb{M}(\Gamma, D)$ if and only if we have for any $x, y \in V$:

(a) $x \prec y$ implies $(\lambda(x), \lambda(y)) \in D$, and (b) $x \parallel y$ implies $(\lambda(x), \lambda(y)) \notin D$.

This characterization leads to the definition of an infinitary extension of traces: A Γ-labeled poset $(V, \leq, \lambda)$ is a *real trace* if V is at most countably infinite, $\downarrow x$ is finite for any $x \in V$, and (a) and (b) hold in $(V, \leq, \lambda)$. The set of real traces over the dependence alphabet (Γ, D) is denoted by $\mathbb{R}(\Gamma, D)$. Note that the infinite product $t_0 * t_1 * t_2 \cdots$ of finite traces $t_i \in \mathbb{M}(\Gamma, D)$ can be naturally defined and yields a real trace.

A set $L \subseteq \mathbb{R}(\Gamma, D)$ of real traces is *recognizable* [9,8] if there exists a finite semigroup $(S, *)$ and a semigroup homomorphism $\eta : \mathbb{M}(\Gamma, D) \to (S, *)$ such that $t_0 * t_1 * t_2 \cdots \in L$ implies $s_0 * s_1 * s_2 \cdots \in L$ for any finite traces $s_i, t_i \in \mathbb{M}(\Gamma, D)$ with $\eta(t_i) = \eta(s_i)$ for $i \in \omega$.

2.3 Monadic Second Order Logic

In this section, we will define monadic second order formulas and their interpretations over Γ-labeled pomsets. *Monadic formulas* involve first order variables $x, y, z \ldots$ for vertices and monadic second order variables $X, Y, Z, \ldots$ for sets of vertices. They are built up from the atomic formulas $\lambda(x) = a$ for $a \in \Gamma$, $x \leq y$, and $x \in X$ by means of the boolean connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$, $\leftrightarrow$ and quantifiers $\exists, \forall$ (both for first order and for second order variables). Formulas without free variables are called sentences. The satisfaction relation $\models$ between Γ-labeled posets $t = (V, \leq, \lambda)$ and monadic sentences φ is defined canonically with the understanding that first order variables range over the vertices of V and second order variables over subsets of V.

Let $\mathcal{C}$ be a set of Γ-labeled posets and φ a monadic sentence. Furthermore, let $L = \{t \in \mathcal{C} \mid t \models \varphi\}$ denote the set of posets from $\mathcal{C}$ that satisfy φ. Then we say that the sentence φ *axiomatizes the set* L *relative to* $\mathcal{C}$ or that L is *monadically axiomatizable relative to* $\mathcal{C}$.

In [6], it was shown that a set of real traces is recognizable if and only if it is monadically axiomatizable relative to the set of all real traces. This result generalizes Büchi's Theorem that states the same fact for ω-words.

3 From ω-Series-Rational to Monadically Axiomatizable Sets

Let $t = (V, \leq, \lambda)$ be some N-free pomset and $X \subseteq V$. Since any antichain in t is finite, the set X is finite if and only if it is bounded by an antichain. Hence the formula $\exists A \forall a, b, x(((a, b \in A \wedge a \leq b) \rightarrow b \leq a) \wedge (x \in X \rightarrow \exists c : (c \in A \wedge x \leq c)))$ expresses that the set X is finite. We denote this formula, that will be useful in the following proof, by finite(X).

Lemma 2. *Let Γ be an alphabet and let $L \subseteq \mathrm{NF}^\infty(\Gamma)$ be an ω-series-rational language. Then L is monadically axiomatizable relative to $\mathrm{NF}^\infty(\Gamma)$.*

Proof. Clearly, any set $\{s\}$ with s finite can be monadically axiomatized. Now let S and T be two sets of N-free pomsets axiomatized by the monadic sentences σ and τ, respectively. Then $S \cup T$ is axiomatized by $\sigma \vee \tau$. The set $S \parallel T$ consists of all N-free pomsets satisfying

$$\exists X(X \neq \emptyset \wedge X^{co} \neq \emptyset \wedge \forall x \forall y(x \in X \wedge y \notin X \rightarrow x \parallel y) \wedge \sigma \restriction X \wedge \tau \restriction X^{co})$$

where $\sigma \restriction X$ is the restriction of σ to the set X and $\tau \restriction X^{co}$ that of τ to the complement of X. The sequential product can be dealt with similarly.

Next we show that S^+ can be described by a monadic sentence: The idea of a sentence axiomatizing S^+ is to color the vertices of an N-free pomset s by two colors such that the coloring corresponds to a factorization in factors $s = s_1 \cdot s_2 \cdot s_3 \cdots s_n$ where every factor s_i belongs to S. The identification of the S-factors will be provided by the property of being a maximal convex one-colored set. More formally, we define $\varphi = \sigma \vee \exists X \exists Y(\varphi_1 \wedge \varphi_X \wedge \varphi_Y \wedge \text{finite}(X) \wedge \text{finite}(Y))$ where φ_1 asserts that X and Y form a partition of the set of vertices such that vertices from X and vertices from Y are mutually comparable. The formula φ_X states that the maximal subsets of X that are convex satisfy σ, i.e.

$$\varphi_X = \forall Z \left[\begin{pmatrix} Z \subseteq X \wedge Z \text{ is convex} \wedge Z \neq \emptyset \wedge \\ \forall Z'(Z \subseteq Z' \subseteq X \wedge Z' \text{ is convex} \rightarrow Z = Z') \end{pmatrix} \rightarrow \sigma \restriction Z \right]$$

and the formula φ_Y is defined similarly with Y taking the place of X. Asserting that the sets X and Y are finite ensures that the sentence φ is satisfied by finite N-free pomsets, only. Hence we get indeed that φ axiomatizes S^+.

To axiomatize S^ω, we can proceed similarly to the case S^+. □

The remaining pages are devoted to the converse of the above theorem, i.e. to the question whether all monadically axiomatizable sets are ω-series-rational. Before we continue, let us sketch an idea how to tackle this problem, and explain, why we will not follow this way. Any N-free pomset is the value of a term over the signature $\{\prod, \cdot, \|\} \cup \Gamma$ (where $\prod$ has arity ω). For a set L of N-free pomsets, let T_L denote the set of all terms over the given signature whose value belongs to L. Note that T_L is a set of trees labeled by the elements from $\{\prod, \cdot, \|\} \cup \Gamma$. Similarly to [2], one can show that T_L is monadically axiomatizable whenever L is monadically axiomatizable. Hence, by the results from [20], T_L is recognizable. This implies that T_L is a rational tree language over the alphabet $\{\prod, \cdot, \|\} \cup \Gamma \cup X$ for some finite set X of additional symbols. Since I do not know how to infer that L is series-rational in case T_L is rational over an extended alphabet, I follow another way in the proof of the converse of the above theorem.

4 From Monadically Axiomatizable to ω-Recognizable Sets

4.1 ω-Recognizable Sets

Recall that a set of infinite words $L \subseteq \Gamma^\omega$ is Büchi-recognizable if there exists a finite semigroup $(S, *)$ and a semigroup homomorphism $\eta : \Gamma^+ \rightarrow (S, *)$ such

that for any $u_i, v_i \in \Gamma^*$ with $\eta(u_i) = \eta(v_i)$ for $i \in \omega$, we have $u_0 u_1 u_2 \cdots \in L$ if and only if $v_0 v_1 v_2 \cdots \in L$ (cf. [17]). Here, we use this characterization as a definition and transfer it into the context of N-free pomsets:

Let S be a set that is equipped with two binary operations $\cdot$ and $\|$. We assume these two operations to be associative and, in addition, $\|$ to be commutative. Then $(S, \cdot, \|)$ is an *sp-algebra*. Note that the set of finite N-free pomsets is an sp-algebra. Mappings between sp-algebras that commute with the two products will be called *sp-homomorphisms*.

Let X be a set of variables that will range over elements of $\mathrm{NF}(\Gamma)$. We call the terms over $\cdot$ and $\|$ that contain variables in X *finite terms*. Now let t_i be finite terms for $i \in \omega$. Then $\prod_{i\in\omega} t_i$ is a *term* and any finite term is a term. Furthermore, if t is a finite term and t_i are terms for $1 \leq i \leq n$, then $t \cdot t_1$ and $t_1 \| t_2 \| \cdots t_n$ are terms, too. Now let $f : X \to \mathrm{NF}(\Gamma)$. Then $f(t) \in \mathrm{NF}^\infty(\Gamma)$ is defined naturally for any term t. Let $L \subseteq \mathrm{NF}^\infty(\Gamma)$. Then L is *ω-recognizable* if there exists a finite sp-algebra $(S, \cdot, \|)$ and an sp-homomorphism $\eta : \mathrm{NF}(\Gamma) \to (S, \cdot, \|)$ such that for any term t and any mappings $f, g : X \to \mathrm{NF}(\Gamma)$ with $\eta \circ f = \eta \circ g$, we have $f(t) \in L$ if and only if $g(t) \in L$. In this case, we will say that the sp-homomorphism η *recognizes* L. In [15], recognizable subsets of $\mathrm{NF}(\Gamma)$ are defined: A set $L \subseteq \mathrm{NF}(\Gamma)$ is *recognizable* if there exists a finite sp-algebra $(S, \cdot, \|)$ and an sp-homomorphism $\eta : \mathrm{NF}(\Gamma) \to (S, \cdot, \|)$ such that $L = \eta^{-1}\eta(L)$. One can easily check that $L \subseteq \mathrm{NF}(\Gamma)$ is recognizable in this sense if and only if it is ω-recognizable.

Example 3. Let $(V, \leq)$ be a tree without maximal elements, such that $\downarrow v$ is finite for any $v \in V$, any node has at most 2 upper neighbors, and almost all nodes from V have only one upper neighbor. Let n be the number of branching points of $(V, \leq)$. Then we call $(V, \leq)$ a *tree with n branching points*. Note that $(V, \leq, \lambda)$ is an N-free pomset.

Now let N be a set of natural numbers and let L_N denote the set of all Γ-labeled trees with n branching points for some $n \in N$. We show that L_N is ω-recognizable:

We consider the sp-algebra $S = \{1, 2\}$ with the mapping $\eta : \mathrm{NF}(\Gamma) \to S$ defined by $\eta(t) = \min(w(t), 2)$ for any $t \in \mathrm{NF}(\Gamma)$. To obtain an sp-homomorphism, let $x \| y = \min(2, x + y)$ and $x \cdot y = \max(x, y)$ for any $x, y \in S$. Now let T be a term and $f, g : X \to \mathrm{NF}(\Gamma)$ with $\eta \circ f = \eta \circ g$. Furthermore, assume $f(T) \in L_N$, i.e. that $f(T)$ is a tree with $n \in N$ branching points. As $f(T)$ has no leaves, every parallel product $\|$ in T is applied to two non-finite terms and similarly the second factor of every sequential product $\cdot$ in T is a non-finite term. Hence every variable x_i (that occurs in T at all) occurs in T either as a left factor of a sequential product $\cdot$ or within the scope of an infinite product $\|$. Since $f(T)$ is a tree, this implies that $f(x_i)$ is a (finite) linear order, i.e. $w(f(x_i)) = 1$. Now $\eta \circ f = \eta \circ g$ implies $w(g(x_i)) = 1$. Hence the N-free pomset $g(T)$ differs from the tree with n branching points $f(T)$ only in some non-branching pieces. Thus, $g(T)$ is a tree with n branching points, i.e. $g(T) \in L_N$ as required. Hence we showed that L_N is indeed ω-recognizable.

By the example above, the number of ω-recognizable subsets of $\mathrm{NF}^\infty(\Gamma)$ is $2^{\aleph_0}$. Since there are only countably many monadic sentences or ω-series-rational languages, not all ω-recognizable sets are monadically axiomatizable or ω-series-rational. Later, we will see that these three notions coincide for width-bounded sets. But first, we show that any ω-recognizable set of N-free pomsets of finite width is of a special form (cf. Proposition 6).

Let $(S, \cdot, \|)$ be an sp-algebra. Then a pair $(s, e) \in S^2$ is *linked* if $s \cdot e = s$ and $e \cdot e = e$. A *simple term* of order 1 is an element of S or a linked pair (s, e). Now let $n > 1$, σ_i for $i = 1, 2, \ldots n$ be simple terms of order n_i, and $s \in S$. Then $s \cdot (\sigma_1 \parallel \sigma_2 \parallel \cdots \sigma_n)$ is a simple term of order $n_1 + n_2 + \ldots n_n$.

For an sp-homomorphism $\eta : \mathrm{NF}(\Gamma) \to S$ and a simple term σ, we define the language $L_\eta(\sigma)$ inductively: If $\sigma \in S$, we set $L_\eta(\sigma) := \eta^{-1}(\sigma)$. For a linked pair (s, e), we define $L_\eta(s, e) := \eta^{-1}(s) \cdot (\eta^{-1}(e))^\omega$. Furthermore, $L_\eta(s \cdot (\sigma_1 \parallel \sigma_2 \parallel \ldots \sigma_n)) := \eta^{-1}(s) \cdot (L_\eta(\sigma_1) \parallel L_\eta(\sigma_2) \parallel \cdots L_\eta(\sigma_n))$.

Lemma 4. *Let Γ be an alphabet, $(S, \cdot, \|)$ a finite sp-algebra and $\eta : \mathrm{NF}(\Gamma) \to (S, \cdot, \|)$ an sp-homomorphism. Let furthermore $t \in \mathrm{NF}^\infty(\Gamma)$ be an N-free pomset of finite width. Then there exist simple terms $\tau_1, \tau_2, \ldots, \tau_m$ of order at most $w(t)$ with $n \leq w(t)$ and $t \in L_\eta(\tau_1) \parallel L_\eta(\tau_2) \parallel \cdots L_\eta(\tau_m)$.*

Proof. If t is finite, the lemma is obvious since the element $s = \eta(t)$ of S is a simple term of order 1. Thus, we may assume t to be infinite. First consider the case that $t = \prod_{i \in \omega} t_i$ is an infinite product of finite N-free pomsets t_i. Let $s_i := \eta(t_i)$. A standard application of Ramsey's Theorem [19] (cf. also [17]) yields the existence of positive integers n_i for $i \in \omega$ and a linked pair $(s, e) \in S^2$ such that $s = s_0 s_1 \cdots s_{n_0}$ and $e = s_{n_i+1} \cdot s_{n_i+2} \cdots s_{n_{i+1}}$ for $i \in \omega$. Hence $t \in L_\eta(s, e)$. Since (s, e) is a simple term of order $1 \leq w(t)$, we showed the lemma for infinite products of finite N-free pomsets.

Now the proof proceeds by induction on the width $w(t)$ of an N-free pomset t. By [1], t is either a parallel product, or an infinite sequential product, or of the form $s \cdot (t_1 \parallel t_2)$ for some finite N-free pomset s. In any of these cases, one uses the induction hypothesis (which is possible since e.g. $w(t_1) < w(t)$ in the third case). □

The following lemma shows that the set $L_\eta(\tau)$ is contained in any ω-recognizable set L that intersects $L_\eta(\tau)$.

Lemma 5. *Let Γ be an alphabet, $(S, \cdot, \|)$ a finite sp-algebra, and $\eta : \mathrm{NF}(\Gamma) \to (S, \cdot, \|)$ an sp-homomorphism and τ_i a simple term for $1 \leq i \leq m$. Let furthermore $t, t' \in L_\eta(\tau_1) \parallel L_\eta(\tau_2) \parallel \cdots L_\eta(\tau_m)$. Then there exist a term T and mappings $f, g : X \to \mathrm{NF}(\Gamma)$ with $\eta \circ f = \eta \circ g$ such that $f(T) = t$ and $g(T) = t'$.*

Proof. First, we show the lemma for the case $m = 1$ (for simplicity, we write τ for τ_1): In this restricted case, the lemma is shown by induction on the construction of the simple term τ. For $\tau = s \in S$, the term $T = x$ and the mappings $f(y) = t$ and $g(y) = t'$ for any $y \in X$ have the desired properties. For a linked pair $\tau = (s, e)$, the term $T = \prod_{i \in \omega} x_i$ can be used to show the statement. For

$\tau = s \cdot (\tau_1 \parallel \tau_2 \parallel \cdots \tau_n)$, one sets $T = x(T_1 \parallel T_2 \parallel \cdots T_n)$ where $x \in X$ and T_i is a term that corresponds to τ_i. We can assume that no variable occurs in T_i and in T_j for $i \neq j$ and that x does not occur in any of the terms T_i. Then the functions f_i and g_i, that exist by the induction hypothesis, can be joint which results in functions f and g satisfying the statement of the lemma. □

Let L be an ω-recognizable set of N-free pomsets of finite width. The following proposition states that L is the union of languages of the form $L_\eta(\tau_1) \parallel L_\eta(\tau_2) \parallel \cdots L_\eta(\tau_m)$. But this union might be infinite. The proof is immediate by Lemmas 4 and 5.

Proposition 6. *Let Γ be an alphabet and $L \subseteq \mathrm{NF}^\infty(\Gamma)$ be a set of N-free pomsets of finite width. Let L be recognized by the sp-homomorphism $\eta : \mathrm{NF}(\Gamma) \to (S, \cdot, \parallel)$, and let $\mathcal{T}$ denote the set of finite tuples of simple terms $(\tau_1, \tau_2, \ldots, \tau_m)$ such that $\emptyset \neq L \cap (L_\eta(\tau_1) \parallel L_\eta(\tau_2) \parallel \cdots L_\eta(\tau_m))$. Then*

$$L = \bigcup_{(\tau_1, \tau_2, \ldots, \tau_m) \in \mathcal{T}} (L_\eta(\tau_1) \parallel L_\eta(\tau_2) \parallel \cdots L_\eta(\tau_m)).$$

4.2 Monadically Axiomatizable Sets of Bounded Width

Now, we concentrate on sets of N-free pomsets whose width is *uniformly* bounded by some integer. It is our aim to show that a set of bounded width that is monadically axiomatizable relative to $\mathrm{NF}^\infty(\Gamma)$ is ω-recognizable.

Let $(V, \leq)$ be a partially ordered set. Following [12], we define a directed graph $(V, E) = \mathrm{spine}(V, \leq)$, called *spine*, of $(V, \leq)$ as follows: The edge relation is a subset of the strict order $<$. A pair (x, y) with $x < y$ belongs to E if either $x \prec y$ or for any $z \in V$ we have $x \prec z \Rightarrow z \leq y$ as well as $z \prec y \Rightarrow x \leq z$. Thus, $(x, y) \in E$ if either $x \prec y$ or $x < y$ and any upper neighbor of x (any lower neighbor of y) is below y (above x, respectively).

Let $\mathrm{maxco}(\mathrm{spine}(V, \leq))$ denote the maximal size of a totally unconnected set of vertices in the graph $\mathrm{spine}(V, \leq)$. The restriction of the following lemma to finite partially ordered sets was shown in [12]. The extension we use here is an obvious variant of this result:

Lemma 7 (cf. [12]). *Let $n > 0$. There exists a dependence alphabet (Γ_n, D_n) with the following property: Let $(V, \leq)$ be a poset with $\mathrm{maxco}(\mathrm{spine}(V, \leq)) \leq n$ such that $\downarrow x$ is finite for any $x \in V$. Then there exists a mapping $\lambda : V \to \Gamma_n$ such that $(V, \leq, \lambda) \in \mathbb{R}(\Gamma_n, D_n)$ is a real trace over the dependence alphabet (Γ_n, D_n).*

We introduced the spine of a partially ordered set and mentioned the result of Hoogeboom and Rozenberg since the spine of an N-free partially ordered set is "small" as the following lemma states.

Lemma 8. *Let Σ be an alphabet and $t = (V, \leq, \lambda) \in \mathrm{NF}^\infty(\Sigma)$ be an N-free pomset of finite width. Then $\mathrm{maxco}(\mathrm{spine}(t)) < 2w(t) \cdot (w(t) + 1)$.*

Let Γ and Σ be two alphabets. A set of Σ-labeled posets L is the *projection* of a set M of Γ-labeled posets if there exists a mapping $\pi : \Gamma \to \Sigma$ such that $L = \{(V, \leq, \pi \circ \lambda) \mid (V, \leq, \lambda) \in M\}$, i.e. L is the set of relabeled (w.r.t. π) posets from M.

Now let L be a set of N-free pomsets of width at most n. Then the two lemmas above show that L is the projection of a set of real traces over a finite dependence alphabet. Because of this relation, we now start to consider sets of N-free real traces:

Languages of N-free real traces. Recall that we want to show that any monadically axiomatizable set of N-free pomsets is ω-recognizable. By [6], any monadically axiomatizable set of (N-free) real traces is recognizable. Therefore, we essentially have to show that any set $L \subseteq \mathbb{R}(\Gamma, D)$ of N-free real traces that is recognizable in $\mathbb{R}(\Gamma, D)$, is ω-recognizable in $\mathrm{NF}(\Gamma)$. Thus, in particular we have to construct from a semigroup homomorphism $\eta : \mathbb{M}(\Gamma, D) \to (S, *)$ into a finite semigroup $(S, *)$ an sp-homomorphism $\gamma : \mathrm{NF}(\Gamma) \to (S^+, \cdot, \|)$ into some finite sp-algebra. This is the content of Lemma 10 that is prepared by the following definition and lemma.

Let $\mathrm{alph}^3(V, \leq, \lambda) := (\lambda \circ \min(V, \leq), \lambda(V), \lambda \circ \max(V, \leq))$ for any finite Γ-labeled poset $(V, \leq, \lambda)$. Let $\mathcal{C}$ be a set of finite Γ-labeled posets (the two examples, we will actually consider, are $\mathcal{C} = \mathrm{NF}(\Gamma)$ and $\mathcal{C} = \mathbb{M}(\Gamma, D)$) and $\eta : \mathcal{C} \to S$ a mapping. Then η is *strongly alphabetic* if it is surjective and if $\eta(t_1) = \eta(t_2)$ implies $\mathrm{alph}^3(t_1) = \mathrm{alph}^3(t_2)$ for any $t_i = (V_i, \leq, \lambda_i) \in \mathcal{C}$. Using the concept of a dependence chain, one can easily show the existence of a semigroup homomorphism η into a finite semigroup such that η is strongly alphabetic:

Lemma 9. *Let (Γ, D) be a dependence alphabet. There exists a finite semigroup $(S, *)$ and a* strongly alphabetic *semigroup homomorphism $\eta : \mathbb{M}(\Gamma, D) \to (S, *)$.*

Lemma 10. *Let (Γ, D) be a dependence alphabet and $\eta : \mathbb{M}(\Gamma, D) \to (S, *)$ be a strongly alphabetic semigroup homomorphism into a finite semigroup $(S, *)$. Then there exists a finite sp-algebra $(S^+, \cdot, \|)$ with $S \subset S^+$ and a strongly alphabetic sp-homomorphism $\gamma : \mathrm{NF}(\Gamma) \to (S^+, \cdot, \|)$ such that*

1. *$\eta(t) = \gamma(t)$ for $t \in \mathbb{M}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$ and*
2. *$\gamma(t) \in S$ implies $t \in \mathbb{M}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$ for any $t \in \mathrm{NF}(\Gamma)$.*

Proof. Since η is strongly alphabetic, there is a function $\mathrm{alph}^3 : S \to (\mathcal{P}(\Gamma) \setminus \{\emptyset\})^3$ with $\mathrm{alph}^3(\eta(t)) = \mathrm{alph}^3(t)$ for any trace $t \in \mathbb{M}(\Gamma, D)$. From the semigroup $(S, *)$ and the function alph^3, we define an sp-algebra $(S_1, \cdot, \|)$ as follows: Let $S_1 = S \dot\cup (\mathcal{P}(\Gamma) \setminus \{\emptyset\})^3$ and extend the function alph^3 to S_1 by $\mathrm{alph}^3(X) = X$ for $X \in (\mathcal{P}(\Gamma) \setminus \{\emptyset\})^3$. Now let

$$x \cdot y = \begin{cases} x * y & \text{if } \pi_3 \circ \mathrm{alph}^3(x) \times \pi_1 \circ \mathrm{alph}^3(y) \subseteq D \\ (\pi_1 \circ \mathrm{alph}^3(x), \pi_2 \circ \mathrm{alph}^3(x) \cup \pi_2 \circ \mathrm{alph}^3(y), \pi_3 \circ \mathrm{alph}^3(y)) & \text{otherwise} \end{cases}$$

$$x \parallel y = \begin{cases} x * y & \text{if } \pi_2 \circ \mathrm{alph}^3(x) \times \pi_2 \circ \mathrm{alph}^3(y) \subseteq I \\ \mathrm{alph}^3(x) \cup^3 \mathrm{alph}^3(y) & \text{otherwise} \end{cases}$$

for any $x, y \in S$ where $\cup^3$ is the componentwise union of elements of $(\mathcal{P}(\Gamma)\setminus\{\emptyset\})^3$. Let furthermore $x \cdot X = X \cdot x = x \parallel X = X \parallel x = X \cup^3 \mathrm{alph}^3(x)$ for any $x \in S_1$ and $X \in (\mathcal{P}(\Gamma) \setminus \{\emptyset\})^3$. One can easily check that the mappings $\cdot$ and $\parallel$ are associative and that the parallel product $\parallel$ is commutative. Now let $\gamma : \mathrm{NF}(\Gamma) \to S_1$ be defined by $\gamma(t) = \eta(t)$ for $t \in \mathbb{M}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$ and $\gamma(t) = \mathrm{alph}^3(t)$ for $t \in \mathrm{NF}(\Gamma) \setminus \mathbb{M}(\Gamma, D)$ and let S^+ be the image of γ. Then γ is a strongly alphabetic sp-homomorphism onto $(S^+, \cdot, \parallel)$. □

Lemma 11. *Let (Γ, D) be a dependence alphabet. Then $\mathbb{R}(\Gamma, D) \cap \mathrm{NF}^\infty(\Gamma)$ is ω-recognizable.*

Proof. By Lemma 9, there exists a strongly alphabetic semigroup homomorphism $\alpha : \mathbb{M}(\Gamma, D) \to (T, *)$ into a finite semigroup $(T, *)$. Then, by Lemma 10, we find a strongly alphabetic sp-homomorphism $\eta : \mathrm{NF}(\Gamma) \to (S, \cdot, \parallel)$ that coincides with α on $\mathbb{M}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$ such that $\eta(t) \in T$ implies $t \in \mathbb{M}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$. Let furthermore $f, g : X \to \mathrm{NF}(\Gamma)$ be functions with $\eta \circ f = \eta \circ g$. By induction on the construction of a term t, one shows that $f(t)$ is a trace if and only $g(t)$ is a trace. □

From a term t, we construct a finite or infinite sequence $\mathrm{lin}(t)$ over X inductively: First, $\mathrm{lin}(x_i) = (x_i)$. Now let t_i for $i \in \omega$ be finite terms. Then $\mathrm{lin}(t_1 \cdot t_2) = \mathrm{lin}(t_1 \parallel t_2)$ is the concatenation of the sequences $\mathrm{lin}(t_1)$ and $\mathrm{lin}(t_2)$. Similarly, $\mathrm{lin}(\prod_{i\in\omega} t_i)$ is the concatenation of the sequences $\mathrm{lin}(t_i)$. Now let t_1, t_2 be terms and t be a finite term. Then $\mathrm{lin}(t \cdot t_1)$ is the concatenation of the sequences $\mathrm{lin}(t)$ and $\mathrm{lin}(t_1)$ (note that $\mathrm{lin}(t)$ is finite). If $\mathrm{lin}(t_1)$ is finite, let $\mathrm{lin}(t_1 \parallel t_2)$ be the concatenation of $\mathrm{lin}(t_1)$ with $\mathrm{lin}(t_2)$ and, if $\mathrm{lin}(t_1)$ is infinite but $\mathrm{lin}(t_2)$ is finite, let $\mathrm{lin}(t_1 \parallel t_2)$ be the concatenation of $\mathrm{lin}(t_2)$ with $\mathrm{lin}(t_1)$. If both $\mathrm{lin}(t_1)$ and $\mathrm{lin}(t_2)$ are infinite, $\mathrm{lin}(t_1 \parallel t_2)$ is the alternating sequence $(y_1^1, y_1^2, y_2^1, y_2^2, \dots)$ with $\mathrm{lin}(t_i) = (y_1^i, y_2^i, \dots)$ for $i = 1, 2$.

For a term t with $\mathrm{lin}(t) = (y_1, y_2, y_3, \dots)$ and a mapping $f : X \to \mathrm{NF}(\Gamma)$, let $\bigstar(f, t) := f(y_1) * f(y_2) * f(y_3) \cdots$ denote the infinite trace product of the pomsets $f(y_i)$. Note that $\bigstar(f, t)$ is a Γ-labeled poset that in general need not be a trace nor an N-free pomset. The following lemma implies that in certain cases $\bigstar(f, t)$ is a real trace. It is shown by induction on the depth of a term.

Lemma 12. *Let (Γ, D) be a dependence alphabet. Let t be a term and $f : X \to \mathrm{NF}(\Gamma)$. If $f(t)$ is a real trace, then $f(t) = \bigstar(f, t)$.*

Now we can show that at least any monadically axiomatizable set of N-free real traces is ω-recognizable:

Proposition 13. *Let (Γ, D) be a dependence alphabet and $\varkappa$ be a monadic sentence. Then the set $L = \{t \in \mathbb{R}(\Gamma, D) \cap \mathrm{NF}^\infty(\Gamma) \mid t \models \varkappa\}$ is ω-recognizable*

Proof. By [6], the set L is a recognizable language of real traces. Hence there is a semigroup homomorphism $\eta : \mathbb{M}(\Gamma, D) \to (S, *)$ into a finite semigroup $(S, *)$ such that, for any sequences x_i, y_i of finite traces with $\eta(x_i) = \eta(y_i)$, we have

$x_1 * x_2 * \cdots \in L$ if and only if $y_1 * y_2 * \cdots \in L$. By Lemma 9, we may assume that η is strongly alphabetic. By Lemma 10, there exists a finite sp-algebra $(S^+, \cdot, \|)$ and a strongly alphabetic sp-homomorphism $\eta^+ : \mathrm{NF}(\Gamma) \to S^+$ that coincides with η on $\mathbb{M}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$.

By Lemma 11, there exists an sp-homomorphism $\delta : \mathrm{NF}(\Gamma, D) \to (T, \cdot, \|)$ that recognizes $\mathbb{R}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$. Now let $\alpha = \eta^+ \times \delta : \mathrm{NF}(\Gamma) \to S \times T$. We show that α recognizes L:

Let t be a term with $\mathrm{lin}(t) = (y_1, y_2, \dots)$ and $f, g : X \to \mathrm{NF}(\Gamma)$ be mappings with $\alpha \circ f = \alpha \circ g$. Suppose $f(t) \in L$. Then $f(t)$ is a real trace. Since $\delta \circ f = \delta \circ g$ and δ recognizes $\mathbb{R}(\Gamma, D) \cap \mathrm{NF}^\infty(\Gamma)$, the Γ-labeled poset $g(t)$ is a real trace, too.

From Lemma 12, we obtain $f(t) = \bigstar(f, t)$ and $g(t) = \bigstar(g, t)$. Since $f(t) \in L$, we have in particular $f(y_1) * f(y_2) * f(y_3) \cdots \in L$. Note that $f(y_i), g(y_i) \in \mathbb{M}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$ and that $\eta^+ \circ f = \eta^+ \circ g$. Since η^+ and η coincide on $\mathbb{M}(\Gamma, D) \cap \mathrm{NF}(\Gamma)$, this implies $\eta(f(y_i)) = \eta(g(y_i))$. Since η recognizes the language of real traces L, we obtain $g(y_1) * g(y_2) * g(y_3) \cdots \in L$. □

Languages of N-free pomsets. Following Lemma 8, we explained that any width-bounded set of N-free pomsets is the projection of a set of N-free real traces over some dependence alphabet. This is the crucial point in the following proof.

Proposition 14. *Let $m \in \omega$, let Σ be an alphabet and φ a monadic sentence over Σ. Then the set $L = \{t \in \mathrm{NF}^\infty(\Sigma) \mid t \models \varphi$ and $w(t) \leq m\}$ is ω-recognizable.*

Proof. Let (Γ_n, D_n) be the dependence alphabet from Lemma 7 with $n = 2m(m+1)$. Now let $\Gamma = \Gamma_n \times \Sigma$ and $((A, a), (B, b)) \in D$ iff $(A, B) \in D_n$ for any $(A, a), (B, b) \in \Gamma$. Let $\Pi(V, \leq, \lambda) = (V, \leq, \pi_2 \circ \lambda)$ be the canonical projection from Γ-labeled posets to the set of Σ-labeled posets and consider the set $K := \{t \in \mathbb{R}(\Gamma, D) \mid \Pi(t) \in L\}$.

By Lemma 8, $\Pi(K) = L$. In the monadic sentence φ, replace any subformula of the form $\lambda(x) = a$ by $\bigvee_{A \in \Gamma_n} \lambda(x) = (A, a)$ and denote the resulting sentence by $\varkappa$. Note that $\varkappa$ is a monadic sentence over the alphabet Γ and that $K = \{t \in \mathbb{M}(\Gamma, D) \mid t \models \varkappa\}$. Since $K \subseteq \mathrm{NF}^\infty(\Gamma)$, we can apply Proposition 13, and obtain that K is ω-recognizable. Now one can show that the class of ω-recognizable languages is closed under projections which gives the desired result. □

5 From ω-Recognizable to ω-Series-Rational Sets

To finish the proof of our main theorem, it remains to show that any ω-recognizable set whose elements have a uniformly bounded width is ω-series-rational. This result is provided by the following proposition:

Proposition 15. *Let Γ be an alphabet and $L \subseteq \mathrm{NF}^\infty(\Gamma)$ be ω-recognizable and width-bounded. Then L is ω-series-rational.*

Proof. Let $\eta : \mathrm{NF}(\Gamma) \to (S, \cdot, \|)$ be an sp-homomorphism that recognizes L. Furthermore, let $n \in \omega$ such that $w(t) \leq n$ for any $t \in L$. Now let $\mathcal{T}$ denote

the set of ($\leq n$)-tuples of simple terms $(\tau_1, \tau_2, \ldots, \tau_k)$ of order at most n such that $\emptyset \neq L \cap (L_\eta(\tau_1) \parallel L_\eta(\tau_2) \parallel \cdots L_\eta(\tau_k))$. Note that $\mathcal{T}$ is finite. The proof of Proposition 6 yields that L is the union of the languages $L_\eta(\tau_1) \parallel L_\eta(\tau_2) \parallel \cdots L_\eta(\tau_k)$ over all tuples from $\mathcal{T}$.

Hence it remains to show that $L_\eta(\tau)$ is ω-series-rational for any simple term τ such that there exists $(\sigma_1, \sigma_2, \ldots, \sigma_k) \in \mathcal{T}$ with $\sigma_i = \tau$ for some $1 \leq i \leq k$. This proof proceeds by induction on the subterms of τ and uses in particular the fact that any width-bounded and recognizable set in $\mathrm{NF}(\Gamma)$ is series-rational [15]. □

Now our main theorem follows from Lemma 2, Propositions 14 and 15.

Theorem 16. *Let Γ be an alphabet and $L \subseteq \mathrm{NF}^\infty(\Gamma)$. Then the following are equivalent:*

1. *L is ω-series-rational.*
2. *L is monadically axiomatizable relative to $\mathrm{NF}^\infty(\Gamma)$ and width-bounded.*
3. *L is ω-recognizable and width-bounded.*

Recall that a set of finite N-free pomsets is recognizable in the sense of [15] if and only if it is ω-recognizable. Therefore, a direct consequence of the main theorem (together with the results from [15,16] where the remaining definitions can be found) is the following

Corollary 17. *Let Γ be an alphabet and $L \subseteq \mathrm{NF}(\Gamma)$ be width-bounded. Then L can be accepted by a branching automaton iff it is recognizable iff series-rational iff monadically axiomatizable.*

References

1. S.L. Bloom and Z. Ésik. Shuffle binoids. *Theoretical Informatics and Applications*, 32:175-198, 1998.
2. B. Courcelle. The monadic second-order logic of graphs. I: Recognizable sets of finite graphs. *Information and Computation*, 85:12-75, 1990.
3. V. Diekert and A. Muscholl. Deterministic asynchronous automata for infinite traces. *Acta Informatica*, 31:379-397, 1994.
4. V. Diekert and G. Rozenberg. The Book of Traces. *World Scientific Publ. Co.*, 1995.
5. M. Droste, P. Gastin, and D. Kuske. Asynchronous cellular automata for pomsets. *Theoretical Comp. Science*, 1999. To appear.
6. W. Ebinger and A. Muscholl. Logical definability on infinite traces. *Theoretical Comp. Science*, 154:67-84, 1996.
7. Z. Ésik and S. Okawa. Series and parallel operations on pomsets. In *Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Comp. Science vol. 1738. Springer, 1999.
8. P. Gastin and A. Petit. Infinite traces. In *[4]*, pages 393-486. 1995.
9. P. Gastin, A. Petit, and W. Zielonka. An extension of Kleene's and Ochmanski's theorems to infinite traces. *Theoretical Comp. Science*, 125:167-204, 1994.

10. J.L Gischer. The equational theory of pomsets. *Theoretical Comp. Science*, 61:199-224, 1988.
11. J. Grabowski. On partial languages. *Ann. Soc. Math. Pol. IV: Fund. Math.*, 4(2):427-498, 1981.
12. H.J. Hoogeboom and G. Rozenberg. Dependence graphs. In *[4]*, pages 43-67. 1995.
13. D. Lapoire. Recognizability equals monadic second order definability, for sets of graphs of bounded tree width. In *STACS'98*, Lecture Notes in Comp. Science vol. 1373, pages 618-628. Springer, 1998.
14. K. Lodaya and P. Weil. A Kleene iteration for parallelism. In V. Arvind and R. Ramanujam, editors, *FST and TCS* 98, Lecture Nodes in Computer Science vol. 1530, pages 355-366. Springer, 1998.
15. K. Lodaya and P. Weil. Series-parallel posets: algebra, automata and languages. In M. Morvan, Ch. Meinel, and D. Krob, editors, *STACS98*, Lecture Nodes in Computer Science vol. 1373, pages 555-565. Springer, 1998.
16. K. Lodaya and P. Weil. Series-parallel languages and the bounded-width property. *Theoretical Comp. Science*, 1999. to appear.
17. D. Perrin and J.-E. Pin. Mots Infinis. Tech. Rep. LITP 93.40, Université Paris 7 (France), 1993. Book to appear.
18. V. Pratt. Modelling concurrency with partial orders. *Int. J. of Parallel Programming*, 15:33-71, 1986.
19. F.P. Ramsey. On a problem of formal logic. *Proc. London Math. Soc.*, 30:264-286, 1930.
20. M. Takahashi. The greatest fixed-points and rational omega-tree languages. *Theoretical Comp. Science*, 44:259-274, 1986.
21. W. Thomas. On logical definability of trace languages. In *V. Diekert, editor, Proc. of the workshop Algebraic Methods in Computer Science, Kochel am See, FRG*, pages 172-182, 1990. Report TUM-I9002, TU Munich.
22. W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. - Informatique Théorique et Applications*, 21:99-135, 1987.

On Deciding if Deterministic Rabin Language Is in Büchi Class

Tomasz Fryderyk Urbański

Institute of Computer Science, University of Warsaw,
ul.Banacha 2, 02-097 Warsaw, Poland
tu153507@zodiac.mimuw.edu.pl

Abstract. In this paper we give a proof that it is decidable for a deterministic tree automaton on infinite trees with Rabin acceptance condition, if there exists an equivalent nondeterministic automaton with Büchi acceptance condition. In order to prove this we transform an arbitrary deterministic Rabin automaton to a certain canonical form. Using this canonical form we are able to say if there exists a Büchi automaton equivalent to the initial one. Moreover, if it is the case, the canonical form allows us also to find a respective Büchi automaton.

AMS Subject Classification 03D05 03B70

1 Introduction

Automata on infinite sequences and infinite trees were first introduced by J.R. Büchi (1962) [1] and M.O. Rabin (1969) [5] in order to find the proof of decidability of monadic logic of one and many successors respectively. Later they have also proved useful in modeling finite state systems whose behaviour can be infinite. They have found applications in describing behaviour of concurrent processes in real time systems or in construction of automatic program verification systems. There exist several classes of finite automata having different expressive power. Among them a meaningful role is played by automata with Büchi and Rabin acceptance conditions (see [6,7]). The theory of these automata gives rise to many natural questions which have not been answered until now. One of these is treated in this paper. It is known that the classes of tree languages recognized by Rabin automata with successive indices induce a proper hierarchy (see [3]). Languages recognized by Büchi automata are located at a low level of this classification. In this context, it is natural to ask if, given some Rabin automaton $\mathcal{A}$ for a language L, we can determine the minimal index of a Rabin automaton recognizing L. For deterministic automata on infinite words this problem has been solved. T.Wilke and H.Yoo [8] gave an algorithm which computes the aforementioned minimal index for any Rabin language consisting of infinite words. At the same time there have been attempts to consider a similar problem for automata on infinite trees in case of some more restricted subclasses of Rabin automata. In the paper [4] by D.Niwiński and I.Walukiewicz there can be found a method of computing the minimal Rabin index for the class

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 663–674, 2000.
© Springer-Verlag Berlin Heidelberg 2000

of infinite tree languages, whose all paths are in some $\omega-$ regular set of infinite words. The problem in general has not been yet solved for automata on infinite trees. In this paper we make one step in this direction, giving a procedure which determines if a language accepted by a deterministic Rabin automaton can be accepted by a (possibly nondeterministic) Büchi automaton. In terms of the hierarchy described below it means determining if the minimal index of a language accepted by a deterministic tree Rabin automaton is equal 1. Our problem is related to the well-known open question of determining the alternation level of a formula of the μ-calculus. Recently M. Otto [9] showed a decision procedure to determine whether a formula is equivalent to an alternation-free formula. It is known that the expressive power of Büchi automata corresponds to that of the formulas of the alternation level $\nu\mu$, however we do not know of any direct μ-calculus characterization of deterministic automata. Nevertheless we may hope that extension of our techniques can help to solve, at least to some extent, the aforementioned μ-calculus problem.

2 Basic Definitions and Notations

2.1 Mathematical Preliminaries

Let ω denote the set of natural numbers. For an arbitrary function f, we denote its domain by $\mathrm{Dom}(f)$. Throughout the paper Σ denotes a finite alphabet. For a set X, X^* is the set of finite words over X, including the empty word ϵ. We denote the length of a word w by $|w|$ ($|\epsilon|$=0). We let X^ω denote the set of infinite words over X. The concatenation wu of words $w \in X^*$ and $u \in X^* \cup X^\omega$ is defined as usual. A word $w \in X^*$ is a prefix of $v \in X^* \cup X^\omega$, in symbols: $w \leq v$, if there exists u such that $v = wu$; the symbols $\geq, >$ etc. have their usual meaning.
An infinite binary tree over Σ is any function $t : \{0,1\}^\star \to \Sigma$. We denote the set of all such functions by T_Σ. An incomplete binary tree over Σ is a function $t : X \to \Sigma$, where $X \subseteq \{0,1\}^\star$ is closed under prefixes, i.e. $w \in X \wedge w' \leq w \Rightarrow w' \in X$ and satisfies the following condition: $\forall w \in X(w0 \in X \wedge w1 \in X) \vee (w0 \notin X \wedge w1 \notin X)$. The set of incomplete binary trees over Σ is denoted by TP_Σ. Note that $T_\Sigma \subseteq TP_\Sigma$. A node $w \in Dom(t)$ is a leaf of t if $w0 \notin Dom(t)$ and $w1 \notin Dom(t)$. We denote the set of leaves of t by $\mathrm{Fr}(t)$. A node which is not a leaf is called the inner node. Let $t \in TP_\Sigma$ and $w \in t$. Then the symbol $t.w$ denotes a subtree of the tree t induced by a node w defined by

$$Dom(t.w) = \{v \mid wv \in Dom(t)\}$$
$$t.w(v) = t(vw), \text{ for } v \in Dom(t.w)$$

A word π from the set $\{0,1\}^\omega$ is a path in a tree $t \in TP_\Sigma$ if all its finite prefixes are in $Dom(t)$. A word π from the set $\{0,1\}^\star$ is a path in the tree $t \in TP_\Sigma$ if $\pi \in Dom(t)$ and $\pi \in Fr(t)$. Throughout this paper whenever we use the symbol π we will mean a path in the sense of the above definitions.
Below we present some notations used throughout this paper for certain subsets of $\{0,1\}^\star$:

$$[w,v] = \{u \in \{0,1\}^\star \mid w \leq u \leq v\}$$
$$(w,v) = \{u \in \{0,1\}^\star \mid w < u < v\}$$
$$\pi_w = \{u \in \{0,1\}^\star \mid w \leq u < \pi\}$$

We write $\exists! x\phi(x)$ to mean that there exists exactly one x satisfying formula $\phi(x)$, similarly a formula $\exists^\infty x\phi(x)$ means that there exist infinitely many x's satisfying $\phi(x)$. A symbol $\dot{\vee}$ will denote exclusive or. For a n-tuple $x = (g_1, ..., g_n)$ we define $x|_i = g_i$.
Now we will introduce some notations concerning directed graphs, used throughout this paper. A cycle S in an arbitrary graph $\mathcal{G}(V,E)$ is a sequence of nodes $a_1, a_2, \ldots, a_k$ such that $k \geq 2$, $a_1 = a_k$, and $(a_i, a_{i+1}) \in E$, for $i \in \{1, \ldots, k-1\}$; it will be denoted by $[a_1, a_2, \ldots, a_k]$. For a cycle $S = [a_1, a_2, \ldots, a_k]$ a notation $\langle S \rangle$ denotes the set $\{a_1, a_2, \ldots, a_k\}$. Given an arbitrary vertex subset X and a cycle S in a graph $\mathcal{G}(V,E)$ we say that the cycle S omits the set X if $\langle S \rangle \cap$ X $= \emptyset$. A connected component of a vertex $p \in V$ with respect to the set $X \subseteq V$ is defined as follows:

$$\mathrm{Con}(X,p) = \{q \in V \mid \exists S - \text{ a cycle in a graph } \mathcal{G} \text{ such that } (\langle S \rangle \cap X = \emptyset \wedge p, q \in \langle S \rangle)\}$$

The set of all connected components with respect to some set X in a graph $\mathcal{G}$ is denoted by $\mathrm{Con}(X, \mathcal{G})$.

2.2 Stopping Automata

Definition 1. A nondeterministic stopping automaton with Rabin acceptance condition *is a 6-tuple* $= \langle Q, \Sigma, q_0, \Delta, \Omega, A \rangle$, *whose components satisfy the following conditions, respectively:*

- $q_0 \in Q$
- $\Delta \subseteq Q \times \Sigma \times (Q \cup A) \times (Q \cup A)$
- $\Omega = \{(L_1, U_1), \ldots, (L_n, U_n), (\emptyset, U_{n+1})\}$ *where* $\forall i \in \{1, \ldots, n+1\}\, L_i, U_i \subseteq Q$
- $Q \cap A = \emptyset$

Q is a finite set of states. A describes a finite set of stopping states disjoint with Q. Ω will be called the set of acceptance conditions. Sometimes when the set of stopping states is empty we will omit the last component in the above 6-tuple and we will not call such an automaton a stopping one.
In this paper there will also appear another type of stopping Rabin automaton, whose initial state is omitted, what we denote by $\langle Q, \Sigma, -, \Delta, \Omega, A \rangle$. If a construction requires giving the initial state, we will use the following notation:

Definition 2. *For an arbitrary stopping automaton* $\mathcal{A}$, $\mathcal{A}[p]$ *denotes a stopping automaton* $\mathcal{A}$ *with the initial state set to* p.

A run of a stopping automaton $\mathcal{A}$ on a tree $t \in T_\Sigma$ is an incomplete tree $r_{t,A} \in TP_{Q \cup A}$ defined by conditions:

- $r_{t,A}(\epsilon) = q_0$

- $\forall w \in (Dom(r_{t,A}) \setminus \mathrm{Fr}(r_{t,A}))\ (r_{t,A}(w), t(w), r_{t,A}(w0), r_{t,A}(w1)) \in \Delta$.
- $\forall w \in \mathrm{Fr}(r_{t,A})\ r_{t,A}(w) \in A$
- $\forall w \in (Dom(r_{t,A}) \setminus \mathrm{Fr}(r_{t,A}))\ r_{t,A}(w) \in Q$

Note that once the automaton reaches a stopping state it stops operating. For a run $r_{t,A}$ and an arbitrary set $X \subseteq \{0,1\}^\star$ a notation $r_{t,A}(X)$ will be used to denote a set $\{r_{t,A}(w) \mid\ w \in X\}$. We also introduce another notation concerning the runs of stopping automata:

$$In(r_{t,A}|\pi) = \{q \mid \exists^\infty\ w < \pi\ (r_{t,A}(w) = q)\ \}$$

We say that a run $r_{t,A}$ is accepting, what we denote by $r_{t,A} \in Acc$, if:

$$\forall \pi\ (\exists i \in \{1, \ldots n+1\}\ (\mathrm{In}(r_{t,A}|\pi) \cap L_i = \emptyset \wedge \mathrm{In}(r_{t,A}|\pi) \cap U_i \neq \emptyset))$$

We reserve the symbol $\mathrm{L}(\mathcal{A})$ for a language accepted by a stopping automaton $\mathcal{A}$ defined by:

$$L(\mathcal{A}) = \{t \in T_\Sigma \mid \exists r_{t,A} \in Acc\}$$

In this paper we assume that all stopping automata in consideration will satisfy the following conditions:

- $\forall a \in \Sigma\ \forall q \in Q\ \exists p, r \in (Q \cup A)\ (q, a, p, r) \in \Delta$
- $\Omega = \{(L_1, U_1), (L_2, U_2), \ldots, (L_n, U_n), (\emptyset, U_{n+1})\}$,where:
  $(\forall i \in \{1 \ldots n\}\ (L_i \neq \emptyset \wedge L_i \cap U_i = \emptyset)) \wedge (U_{n+1} \cap \bigcup_{i=1}^{n} (L_i \cup U_i) = \emptyset)$
- $\forall x \in \bigcup_{i=1}^{n} L_i \cup U_{n+1}\ \exists t \in T_\Sigma\ \exists r_{t,A} \in Acc\ \exists w \in \{0,1\}^\star\ r_{t,A}(w) = x$

Observe that the first dependency guarantees the existence of a run of a stopping automaton over every tree. States which satisfy the last condition are said to be reachable in some accepting run. It is easy to see that the imposed conditions do not diminish the expressive power of the stopping automata. We would like to emphasize that sets U_i for $i \in \{1, \ldots, n+1\}$ can be empty. The index of a stopping automaton $\mathcal{A}$ is denoted by:

$$Ind(\mathcal{A}) = \begin{cases} 2n+1 & if\ U_{n+1} \neq \emptyset \\ 2n & if\ U_{n+1} = \emptyset \end{cases}$$

We write $\mathrm{Ind}(\Omega)$ instead of $\mathrm{Ind}(\mathcal{A})$, where Ω is the set of acceptance conditions of a stopping automaton $\mathcal{A}$. Moreover for the elements of the set Ω we use the subsequent notation:

$$\forall X = (L_i, U_i) \in \Omega\ \ (X|_1 = L_i\ \wedge X|_2 = U_i)$$

Definition 3. *Stopping automata with index 1 are called* **Büchi automata**.

Languages accepted by some stopping automaton with Rabin (Büchi) acceptance condition are called Rabin (Büchi) languages. It is worth noting that classes of languages recognized by automata with Rabin acceptance condition having successive indices induce a proper hierarchy (see [3]).

Definition 4. *A stopping automaton $\mathcal{A}$ is* **deterministic** *if it satisfies the following condition:*

$$\forall q \in Q\ \forall q', q'', p', p'' \in (Q \cup A)\ \forall a \in \Sigma\ (\ (q, a, q', q'') \in \Delta \wedge (q, a, p', p'') \in \Delta \Rightarrow \\ \Rightarrow q' = p' \wedge q'' = p''\ )$$

Note that if we deal with deterministic automata then we can assume that there exists exactly one run of such an automaton over some tree.

Definition 5. *We say that a stopping automaton $\mathcal{A}$ is* **frontier deterministic** *if the following dependency holds true:*

$$\forall t \in T_\Sigma\ \forall r_{t,A}, s_{t,A}\ (\ Fr\ (r_{t,A}) = Fr(s_{t,A})\ \wedge (\ \forall w \in Fr(r_{t,A})\ r_{t,A}(w) = s_{t,A}(w)))$$

Consequently, frontier determinism guarantees that for some complete tree all possible runs stop in the same nodes reaching the same states.
We will use the notation $\mathcal{A} \simeq \mathcal{B}$ to denote **equivalence** between frontier deterministic stopping automata $\mathcal{A}$ and $\mathcal{B}$. This concept is expressed formally as follows:

- $L(\mathcal{A}) = L(\mathcal{B})$
- $\forall t \in T_\Sigma\ \forall r_{t,A}, s_{t,B}\ (\mathrm{Fr}(r_{t,A}) = \mathrm{Fr}(s_{t,B})\ \wedge\ \forall\, w \in \mathrm{Fr}(r_{t,A})\ r_{t,A}(w) = s_{t,B}(w))$

Observe that the second condition refers to all runs of automata $\mathcal{A}$ and $\mathcal{B}$, not only to the accepting ones.
Set $X \subseteq Dom(t)$ is an **antichain** with respect to the order relation $\leq$ if any two elements of X are incomparable. Let f be a function associating trees $t(w) \in T_\Sigma$ with elements w from X. Then we denote the **substitution** by $t[f]$, the **limit** of the sequence t_n by $\lim t_n$ and the **iteration** of t along the interval $[v, w]$ by $t^{[v,w]}$. Definitions of the above concepts and also the concept of the **trace of iteration** are well known and can be found e.g. in the paper by Niwinski [3]

3 Relevant States

Let $\mathcal{A}$=$\langle Q, \Sigma, q_0, \Delta, \Omega, A\rangle$ denote an arbitrary stopping automaton, where: $\Omega = \{(L_1, U_1), (L_2, U_2), \ldots, (L_n, U_n), (\emptyset, U_{n+1})\}$.

Definition 6. *We say that some state $q \in \bigcup_{i=1}^{n} L_i$ is* **irrelevant in a run** $r_{t,A}$ *if its occurrences are covered by a finite number of paths in this run:*

$$\exists\, \pi_1, \ldots, \pi_k \in \{0, 1\}^\omega\ (r_{t,A}(w) = q \Rightarrow \exists\, i \in \{1, \ldots, k\}\ w < \pi_i) \qquad (1)$$

Furthermore we say that a state $q \in \bigcup_{i=1}^{n} L_i$ is **irrelevant** *for an automaton $\mathcal{A}$ if its occurrences are irrelevant in an arbitrary accepting run of this automaton.*

A state which is not irrelevant is called **relevant**. Observe that if a state p is irrelevant for a stopping automaton $\mathcal{A}$ then for an arbitrary tree $t \in L(\mathcal{A})$ and an arbitrary accepting run $r_{t,A}$ there exists a natural number K such that:

$$\forall\, w \in \{0,1\}^\star (r_{t,A}(w) = p \wedge |w| \geq K) \Rightarrow \exists! \pi\, (w < \pi \wedge p \in In(r_{t,A}|\pi)) \qquad (2)$$

Less formally, if in some node below level K the automaton reaches a state p then this state must occur infinitely often on some uniquely determined path going through this node.

Definition 7. *A* ***gadget*** *of states p and q admitted by an automaton $\mathcal{A}$ is a 4-tuple $G(p,q) = (g_1, g_2, g_3, g_4)$ with $g_i \in \{0,1\}^\star$ satisfying the following conditions:*

- $g_1 < g_2 < g_3 \;\wedge\; g_1 < g_2 < g_4$
- $g_3 \not\leq g_4 \;\wedge\; g_3 \not\geq g_4$
- $\exists\, t \in L(\mathcal{A})\, \exists r_{t,A} \in Acc\, \exists i \in \{1,\ldots,n\}\, (\, r_{t,A}(g_1) = p \;\wedge\; r_{t,A}(g_4) = q \;\wedge$ $r_{t,A}(g_2) = r_{t,A}(g_3) = u \in U_i \;\wedge\; r_{t,A}([g_2,g_3]) \cap\; L_i = \emptyset\,)$
- $r_{t,A}([g_1,g_3]) \cap U_{n+1} = \emptyset \;\wedge\; r_{t,A}([g_1,g_4]) \cap U_{n+1} = \emptyset$

¿From now on when we say that an automaton $\mathcal{A}$ admits a gadget G in a tree t in a run $r_{t,A}$, we mean that t and $r_{t,A}$ satisfy the two conditions above. Note that in this case the language accepted by the automaton $\mathcal{A}$ includes also the iteration of the tree t along the interval $[g_2, g_3]$.

Definition 8. *An automaton $\mathcal{A}$ has* **the accessibility property** *iff*

$$\forall p, p' \in \bigcup_{i=1}^{n} L_i\, \exists t \in L(\mathcal{A}[p])\, \exists r_{t,A[p]} \in Acc\;\; \exists w \in \{0,1\}^\star\, (r_{t,A[p]}(w) = p' \;\wedge\; w \neq \epsilon)$$

Clearly, if $Ind(\mathcal{A}) = 1$ then $\bigcup_{i=1}^{n} L_i = \emptyset$ and therefore $\mathcal{A}$ has the accessibility property. It is easy to prove that if an automaton $\mathcal{A}$ has the accessibility property, admits a gadget $G(p,q)$ for some $p, q \in \bigcup_{i=1}^{n} L_i$ and $Ind(\mathcal{A})$ is even (hence $U_{n+1} = \emptyset$), then it actually admits a gadget $G(p,q)$ for any $p, q \in \bigcup_{i=1}^{n} L_i$. The subsequent lemma characterizes relevantness of states using the notion of a gadget.

Lemma 1. *Let $\mathcal{A}$ be a frontier deterministic stopping automaton of the form $\langle Q, \Sigma, q, \Delta, \Omega, A\rangle$, whose index is even. Then the initial state q is relevant for the automaton $\mathcal{A}$ if and only if this automaton admits a gadget $G(q,q)$.*

If the automaton $\mathcal{A}$ admits a gadget $G(q,q)$ then using the iteration it is easy to obtain a tree accepted by $\mathcal{A}$ in which the occurrences of the state q are relevant. The reverse implication is only slightly more difficult - for the details see [10]. Presently we will prove a lemma, which allows us to transform Rabin's automaton into Büchi form if we impose on all states in the set $\bigcup_{i=1}^{n} L_i$ some condition, which uses the notion of relevantness.

Lemma 2. *Let $\mathcal{A}$ be a deterministic stopping automaton of the form $\langle Q, \Sigma, q, \Delta, \Omega, A\rangle$, where $\Omega = \{(L_1, U_1), \ldots, (L_n, U_n), (\emptyset, U_{n+1})\}$ and $n \geq 1$. If for any state $s \in \bigcup_{i=1}^{n} L_i$ s is irrelevant for an automaton $\mathcal{A}[s]$, then there exists frontier deterministic stopping automaton $\mathcal{B} = \langle Q', \Sigma, q', \Delta', \Omega', A\rangle$ with index 1 equivalent to the automaton $\mathcal{A}$.*

For the proof see [10].

4 Canonical Decomposition of Deterministic Automaton

4.1 On Representation of Automata in Form of a Composition

Consider a sequence of stopping automata $(\mathcal{A})_{i\in\{1,\ldots,k\}}$ defined by dependencies:

$$\forall i \in \{1, \ldots, k\}\ \mathcal{A}_i = \langle Q_i, \Sigma, p_i, \Delta_i, \Omega_i, A_i\rangle \tag{3}$$

$$\forall i \in \{1, \ldots, k\}\ \Omega_i = \{(L_1^i, U_1^i), \ldots, (L_{n_i}^i, U_{n_i}^i), (\emptyset, U_{n_i+1}^i)\}. \tag{4}$$

$$\forall i, j \in \{1, \ldots, k\}\ (i \neq j \Rightarrow Q_i \cap Q_j = \emptyset) \tag{5}$$

We define an automaton denoted by a symbol $\mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_k$ as a stopping automaton $\langle Q', \Sigma, -, \Delta', \Omega', A'\rangle$, whose construction is presented below:

- $Q' = \bigcup_{i=1}^{k} Q_i$
- $\Delta' = \bigcup_{i=1}^{k} \Delta_i$
- $A' = (\bigcup_{i=1}^{k} A_i) \setminus Q'$

The set Ω' is a little more tedious to define and comes as a result of the following construction: Let $m = \max\{ n_i \mid i \in \{1, \ldots, k\}\} + 1$. We change each set Ω_i into a sequence consisting of its elements keeping the same notation: $\Omega_i(j)$ for $j \in \{1, \ldots, n_i + 1\}$, taking care that the only pair of the form $(\emptyset, U)$ is at the position $n_i + 1$. Now we define a sequence $\Omega'(j)$ for $j \in \{1, \ldots, m\}$:

$$\Omega'(j) = \begin{cases} (\bigcup_{\substack{1\leq l\leq k\\ j\leq n_l}} (\Omega_l(j)|_1),\ \bigcup_{\substack{1\leq l\leq k\\ j\leq n_l}} (\Omega_l(j)|_2)\ ), \text{ for } j < m \\ (\emptyset, (\bigcup_{1\leq l\leq k} (\Omega_l(n_l+1)|_2))) \qquad \text{for } j = m \end{cases} \tag{6}$$

As can be seen, we sum the conditions pointwise except for the last pairs, which summed separately form the last condition. Now let Ω' be a set of the elements of the sequence $\Omega'(i)$ for $i \in \{1, \ldots, m\}$. Note that the above construction is not unique. Transforming the set Ω_i into a sequence we can arrange its elements in many different ways. This ambiguity however will have no influence on the proofs and can be easily removed, if we define the set of acceptance conditions of a stopping automaton as a sequence.

Definition 9. *The automaton $\mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_k$ described above will be called* **the composition of a sequence** $(\mathcal{A})_{i\in\{1,\ldots,k\}}$.

Note that the following inequalities hold true:

$$\max\{Ind(\Omega_i) \mid 1 \leq i \leq k\} \leq Ind(\Omega') \leq \max\{Ind(\Omega_i) \mid 1 \leq i \leq k\} + 1 \quad (7)$$

Note that the composition is a Büchi automaton if so is each automaton $\mathcal{A}_i$. For a particular composition $\mathcal{B} = \mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_k$ we define a function **SwitchingStates**^{B} , which associates with each automaton $\mathcal{A}_i = \langle Q_i, \Sigma, -, \Delta_i, \Omega_i, A_i\rangle$ some subset of Q_i in the following way:

$$\text{SwitchingStates}^B(\mathcal{A}_i) = Q_i \cap \bigcup_{j=1}^{k} A_j$$

The function SwitchingStates^B describes, which states of the given automaton are the stopping states of the other components of the composition. In this paper often compositions of the form $\mathcal{B} = \mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_k$ (where $\mathcal{A}_i$ are defined as in conditions 3,4) will satisfy additional conditions, stated below.

Definition 10. *Assume that there exists a partition of a set $\{1, \ldots, k\}$ into two sets X, Y, which satisfy the conditions:*

- $\forall i \in X\ Ind(\mathcal{A}_i) = 1$
- $\forall t \in T_\Sigma \forall r_{t,B} \forall\ \pi \in \{0,1\}^\omega\, (\exists i \in Y\, (In(r_{t,B}|\pi) \cap Q_i \neq \emptyset\ \wedge In(r_{t,B}|\pi) \not\subseteq Q_i))$ $\Rightarrow\ (\exists i \in X\ (In(r_{t,B}|\pi) \cap U_1^i \neq \emptyset))$
- $\forall i \in Y\ (\ Ind(\mathcal{A}_i) > 1\ \Rightarrow SwitchingStates^B(\mathcal{A}_i) = \bigcup_{j=1}^{n_i} L_j^i\ )$
- $\forall i \in Y$ *SwitchingStates*$^B(\mathcal{A}_i)$ *are reachable in the accepting runs of the automaton* $\mathcal{B}$

Composition of a sequence of stopping automata is proper *if there exists a partition X, Y such that the above conditions hold true.*

Intuitively the second condition means: if in some run (not necessarily accepting) on some path the automaton $\mathcal{B}$ visits states of some automaton from the group Y infinitely often, but will never remain in its states forever, then the acceptance conditions are satisfied with regard to some automaton from the group X.
Again assume that $\mathcal{B} = \mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_k = \langle Q, \Sigma, -, \Delta, \Omega, A\rangle$. Let us fix for some $i \in \{1, \ldots, k\}$ an automaton $\mathcal{A}_i = \langle Q_i, \Sigma, -, \Delta_i, \Omega_i, A_i\rangle$. Moreover let there exist an automaton $\mathcal{C} = \langle Q', \Sigma, -, \Delta', \Omega', A'\rangle$ such that $Q' \cap Q = \emptyset$ and a function $f : Q_i \overset{1-1}{\to} Q'$. Then for an arbitrary $q \in Q$ **a substitution** $\mathcal{C} \overset{f}{\mapsto} \mathcal{A}_i$ **into the composition** $\mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_k\,[q]$ is a composition $\mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_{i-1} \oplus \mathcal{C} \oplus \mathcal{A}_{i+1} \oplus \ldots \oplus \mathcal{A}_k[q']$, which we change in the following manner:

- if $q \in Q_i$, we take $q' = f(q)$, otherwise $q' = q$,
- all occurrences of states p from the set SwitchingStates$^{\mathcal{B}}(\mathcal{A}_i)$ are replaced by states $f(p)$

4.2 Automaton Graph

Let $\mathcal{A} = \langle Q, \Sigma, q_0, \Delta, \Omega, A\rangle$ denote an arbitrary stopping automaton such that $\Omega = \{(L_1, U_1), (L_2, U_2), \ldots, (L_n, U_n), (\emptyset, U_{n+1})\}$, where: $n \geq 1$. We will construct **a directed automaton graph** $\mathcal{A}$, which will be denoted by $\mathcal{G}_{\mathcal{A}}$. Initially, we define a directed graph: $\mathcal{H}_{\mathcal{A}} = \langle V, E\rangle$, where $V = \bigcup_{i=1}^{n} L_i \cup U_{n+1}$ and $E \subseteq V \times V$ has the following form:

$$E = \{(x, y) \in V \times V \mid \exists\, t \in L(\mathcal{A})\ \ \exists r_{t,A} \in Acc\, \exists w, w' \in \{0,1\}^{\star}$$
$$(r_{t,A}(w) = x \wedge\ r_{t,A}(w') = y\ \ \wedge w < w'\ \wedge\ r_{t,A}((w, w')) \cap V = \emptyset)\}$$

The above graph will be helpful in defining a directed stopping automaton graph $\mathcal{A}$, $\mathcal{G}_{\mathcal{A}} = \langle$V',E'$\rangle$. Namely, we define:

$$\text{V'} = \ \text{Con}(U_{n+1}, \mathcal{H}_{\mathcal{A}}) \cup \{\ \{q\} \mid \text{Con}(U_{n+1}, q) = \emptyset\ \wedge\ q \in V\}\ ,$$
$$\text{E'} = \{(X, Y) \in V' \times V' \mid \exists x \in X\ \exists y \in Y\ (x, y) \in E\},$$

Let us note that considering the character of the above construction and a condition $U_{n+1} \cap \bigcup_{i=1}^{n} L_i = \emptyset$, which holds true, the following dependency is satisfied:

$$\forall X, Y \in V'\ (X \cap Y = \emptyset) \wedge \forall X \in V'\ \ (X \cap U_{n+1} = \emptyset \vee X \cap \bigcup_{i=1}^{n} L_i = \emptyset)$$

4.3 Canonical Form of Rabin Automaton

Lemma 3. *Consider an arbitrary deterministic stopping automaton $\mathcal{A}$ with index greater than 1, $\langle Q, \Sigma, -, \Delta, \Omega, A\rangle$, where $\Omega = \{(L_1, U_1), \ldots, (L_n, U_n), (\emptyset, U_{n+1})\}$. Moreover we require that if its index is even, the automaton does not have the accessibility property. Then there exists a sequence $(\mathcal{B})_{i\in\{1,\ldots,k\}}$ ($k \geq 2$) of deterministic stopping automata whose composition $\mathcal{B} = \mathcal{B}_1 \oplus \ldots \oplus \mathcal{B}_k = \langle Q', \Sigma, -, \Delta', \Omega', A'\rangle$, is proper and satisfies conditions:*

- *$\forall i \in \{1, \ldots, k\}\ \mathcal{B}_i = \langle Q_i, \Sigma,\ s_i, \Delta_i, \Omega_i, A_i\rangle$ has the accessibility property,*
- *$\forall i \in \{1, .., k\}$ $Ind(\mathcal{B}_i) \leq Ind\ (\mathcal{A})$ (if $Ind(\mathcal{A})$ is odd, $Ind(\mathcal{B}_i) < Ind\ (\mathcal{A})$),*
- *$\exists f : Q \overset{1-1}{\rightarrow} Q'\ (\forall\, p \in Q\ (\mathcal{A}[p] \simeq \mathcal{B}[f(p)]))$*

Moreover, if $\Omega' = \{(L'_1, U'_1), \ldots, (L'_m, U'_m), (\emptyset, U'_{m+1})\}$, then the function f from the last condition satisfies a dependency:

$$\bigcup_{j=1}^{m} L'_j \subseteq f(\bigcup_{i=1}^{n} L_i). \tag{8}$$

$$(\ p \in f(\bigcup_{i=1}^{n} L_i) \setminus \bigcup_{j=1}^{m} L'_j\ \wedge\ \exists\, i \in \{1, \ldots, k\}\ f(p) \in \mathcal{B}_i\ ) \Rightarrow Ind(\mathcal{B}_i) = 1 \tag{9}$$

For the proof see [10].

Lemma 4. *Let* $\mathcal{A}=\mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_k = \langle Q, \Sigma, q, \Delta, \Omega, A\rangle$*, where* $\Omega = \{(L_1, U_1), \ldots, (L_n, U_n), (\emptyset, U_{n+1})\}$*, be a proper composition of frontier deterministic stopping automata. Assume that there exists* $i \in \{1, \ldots, k\}$ *such that the automaton* $\mathcal{A}_i$ *is of the form :* $\langle Q_i, \Sigma, -, \Delta_i, \Omega_i, A_i\rangle$ *and its index is greater than 1. Let:* $\Omega_i = \{(L^i_1, U^i_1), \ldots, (L^i_{n_i}, U^i_{n_i}), (\emptyset, U^i_{n_i+1})\}$*. Suppose furthermore that there exists a proper composition of frontier deterministic stopping automata* $\mathcal{B} = \mathcal{B}_1 \oplus \ldots \oplus \mathcal{B}_l = \langle Q', \Sigma, -, \Delta', \Omega', A'\rangle$ *and a one-to-one function* f *from* Q_i *to* Q'*, which satisfies the two conditions 8, 9 and such that for each* $p \in Q_i$ *the automaton* $\mathcal{A}_i[p]$ *is equivalent to the automaton* $\mathcal{B}[f(p)]$*. Then the substitution* $\mathcal{B} \overset{f}{\mapsto} \mathcal{A}_i$ *into the composition* $\mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_k[q]$ *is proper and equivalent to the composition* $\mathcal{A}$*.*

For the proof see [10].

Theorem 1. *For an arbitrary deterministic stopping automaton* $\mathcal{A}$ *with the index greater than 1 there exist two sequences of stopping automata having the accessibility property* $(\mathcal{A}_i)_{i\leq l}$ *and* $(\mathcal{B}_j)_{j\leq k}$*, where* $l \cdot k \neq 0$*, such that the composition* $\mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_l \oplus \mathcal{B}_1 \oplus \ldots \oplus \mathcal{B}_k[q']$ *is equivalent to the automaton* $\mathcal{A}$ *for some specifically chosen state* q'*. Furthermore, this composition satisfies the following conditions:*

1. *for each* $i \in \{1, \ldots, l\}$ *the automaton* $\mathcal{A}_i$ *is frontier deterministic and its index is 1*
2. *for each* $i \in \{1, \ldots, k\}$ *the automaton* $\mathcal{B}_i$ *is deterministic, and its index is even*
3. $Ind(\mathcal{B}_1) \leq \ldots \leq Ind(\mathcal{B}_k)$
4. *for each* $i \in \{1, \ldots, k\}$ $\mathcal{B}_i = \langle\ Q_i, \Sigma, -, \Delta_i, \Omega_i, A_i\rangle$*, where* $\Omega_i = \{(L^i_1, U^i_1), \ldots, (L^i_{n_i}, U^i_{n_i})\}$*, and for any* $p, r \in \bigcup_{j=1}^{n_i} L^i_j$ *the automaton* $\mathcal{B}_i[p]$ *admits a gadget* $G(p, r)$*.*

Moreover, we can assume that the sequences $(\mathcal{A}_i)_{i\leq l}$ *and* $(\mathcal{B}_j)_{j\leq k}$*, satisfy an additional condition:* $l \leq 1$ *and* $k \leq |\bigcup_{i=1}^{n} L_i|$*.*
The representation of the automaton $\mathcal{A}$ *in the form of a composition satisfying the above conditions will be called* **the canonical form**.

Proof. If the index of the automaton $\mathcal{A}$ is odd or it is even, but the automaton does not have the accessibility property then according to the lemma 3 there exists a sequence of deterministic stopping automata having the accessibility property $(\mathcal{C}_i)_{i\leq m}$, whose composition $\mathcal{C} = \mathcal{C}_1 \oplus \ldots \oplus \mathcal{C}_m[r]$ is equivalent to the automaton $\mathcal{A}$ for some state r. Assume that we can find in the set $\{1, \ldots, m\}$ such j that $\mathrm{Ind}(\mathcal{C}_j)$ is odd and greater than 1. If we again use the lemma 3, this time for the automaton $\mathcal{C}_j$, and obtain a composition $\mathcal{D} = \mathcal{D}_1 \oplus \ldots \oplus \mathcal{D}_{m'}$ and a function f, which establishes equivalence between $\mathcal{C}_j$ and the composition $\mathcal{D}$, then by lemma 4 we can construct the substitution $\mathcal{D} \overset{f}{\mapsto} \mathcal{C}_j$ into the composition $\mathcal{C}$. Thus we obtain a new proper composition, whose all elements have the accessibility property and which is equivalent to the automaton $\mathcal{A}$. This procedure

can be repeated as long as among the components of the composition there are automata, whose index is even and greater than 1. Let us note however that this can be done only finitely many times since according to the lemma 3 each time we introduce into the composition automata with smaller indices than the index of the replaced automaton. Eventually we obtain a sequence $(\mathcal{E}_i)_{i \leq m''}$ of deterministic stopping automata having the accessibility property, whose index is either even or equal 1 and whose composition is equivalent to the automaton $\mathcal{A}$. Now let us take an arbitrary automaton from the above composition $\mathcal{E}_i$ with an even index (if there is such). Assume $\mathcal{E}_i = \langle Q', \Sigma, -, \Delta', \Omega', A' \rangle$, where $\Omega' = \{(L'_1, U'_1), \ldots, (L'_{n'}, U'_{n'})\}$. If for each $p \in \bigcup_{j=1}^{n'} L'_j$ the state p is irrelevant for the automaton $\mathcal{E}_i[p]$ then by lemma 2 there exists a frontier deterministic stopping automaton with index 1 equivalent to $\mathcal{E}_i$ (the construction from the lemma 2 does not depend on the initial state of the initial automaton, so there exists a function f establishing an equivalence between the above automata) Note that a single-element sequence consisting of a Büchi automaton $\mathcal{E}'$ forms a proper composition and we can use the lemma 4 constructing a substitution $\mathcal{E}' \overset{f}{\mapsto} \mathcal{E}_i$ into the composition $\mathcal{E}_1 \oplus \ldots \oplus \mathcal{E}_{m''}$. We continue this procedure as long as it is possible. Next we split the sequence $(\mathcal{E}_i)_{i \leq m''}$ into two subsequences $(\mathcal{A}_i)_{i \leq l}$ and $(\mathcal{B}_j)_{j \leq k}$, putting into the first one automata with index 1 and to the latter one the automata with an even index, thus we obtain the representation of the automaton $\mathcal{A}$ in the form of the composition satisfying the conditions of the lemma. Let us observe that we can compute the composition of all component automata with index 1 and satisfy the condition $l \leq 1$. Moreover let us note that the fourth condition also holds. We know that for any $i \in \{1, \ldots, k\}$ there exists $p(i) \in \bigcup_{j=1}^{n_i} L^i_j$ such that the state $p(i)$ is relevant for the automaton $\mathcal{B}_i[p(i)]$. It follows from the lemma 1 that the automaton $\mathcal{B}_i[p(i)]$ admits a gadget $G(p(i), p(i))$. However since the automaton $\mathcal{B}_i[p(i)]$ has the accessibility property then it also admits gadgets of the form $G(p(i), q)$ for any $q \in \bigcup_{j=1}^{n_i} L^i_j$. Finally, by the definition of the accessibility property and the fact that the index of the automaton $\mathcal{B}_i$ is even, the fourth condition is satisfied. The remaining parts of the thesis are easy observations.

5 Main Results

Theorem 2. *Let $\mathcal{B} = \mathcal{A}_1 \oplus \ldots \oplus \mathcal{A}_l \oplus \mathcal{B}_1 \oplus \ldots \oplus \mathcal{B}_k$ be a canonical form of the deterministic stopping tree automaton $\mathcal{A}$ $l, k \geq 0$. Then $L(\mathcal{A})$ is in Büchi class if and only if $k = 0$.*

For the proof see [10].

Theorem 3. *For an arbitrary deterministic stopping automaton $\mathcal{A}$ we are able to decide if the language accepted by it is in Büchi class.*

Sketch of the proof. By the preceding theorem to prove the decidability of the considered problem we need to show that we can find a canonical form of an arbitrary deterministic stopping automaton. Therefore it suffices to prove that we are able to give a procedure deciding if a given state is irrelevant for some deterministic stopping automaton $\mathcal{A}$. Let us observe that if we had a procedure determining if a state is irrelevant for an automaton whose set of stopping states is empty, then our task would be completed. It follows from the fact that we can transform any stopping automaton into one of the above form. To do this we add to the sets of states a new state x, replace all stopping states by this state and finally add transitions (x, a, x, x) for any $a \in \Sigma$ and a condition $(\emptyset, \{x\})$ to the set of conditions. Thus constructed automaton with the empty set of stopping states has the following property: states which are irrelevant for it are also irrelevant for the initial automaton and vice versa.
Additionally, we have to be able to construct an automaton graph $\mathcal{G}_{\mathcal{A}}$. The proof of decidability of both of the problems is simple and uses the celebrated Rabin theorem on decidability of S2S logic [2]. For a complete proof see [10].

References

1. J.R. Büchi *On a decision method in restricted second order arithmetic*, in: Logic, Methodology and Philosophy of Science, Proc. (1960) International Congr. (1962), 1-11.
2. An.A. Muchnik *Igry na bieskoniecznych dieriewiach i awtomaty s tupikami. Nowoje dokazatielstwo razrieszimosti monadiczieskoj tieorii dwuch sliedowanij*, in: Siemiotika (1985), 16-40.
3. D. Niwiński *Fixed point characterization of infinite behavior of finite-state systems*, Fundamental Study, Theoretical Computer Science 189 Elsevier (1997), 1-69.
4. D. Niwiński I. Walukiewicz *Relating hierarchies of word and tree automata* , Proceedings of "15th Annual Symposium on Theoretical Aspects of Computer Science" (STACS), Paris, France (1998), 320-331.
5. M.O. Rabin *Decidability of second-order theories and automata on infinite trees*, Trans. Amer. Math. Soc. 141, (1969), 1-35.
6. W. Thomas *Automata on infinite objects*, in: Handbook of Theoretical Computer Science, Elsevier (1990) 133-191.
7. W. Thomas *Languages, automata and logic*, Handbook of Formal Languages, Vol.3, Springer-Verlag (1997).
8. T. Wilke, H. Yoo *Computing the Rabin index of a regular language of infinite words*, Information and Computation Vol.130 (1996), 61-70.
9. M. Otto, *Eliminating Recursion in the mu-Calculus*, in: Proceedings of STACS'99, LNCS 1563, Springer 1999, 531-540.
10. T.F.Urbański, *On decidability of some properties of finite automata on infinite trees*, Master of Sci. Thesis, Warsaw University, 1998 (in Polish), English version `http://zls.mimuw.edu.pl/~niwinski/Prace/TU.ps.gz`.

On Message Sequence Graphs and Finitely Generated Regular MSC Languages

Jesper G. Henriksen[1], Madhavan Mukund[2], K. Narayan Kumar[2], and P.S. Thiagarajan[2]

[1] BRICS, University of Aarhus, Denmark.
gulmann@brics.dk
[2] Chennai Mathematical Institute, Chennai, India.
{madhavan,kumar,pst}@smi.ernet.in

Abstract. Message Sequence Charts (MSCs) are an attractive visual formalism widely used to capture system requirements during the early design stages in domains such as telecommunication software. A standard method to describe multiple communication scenarios is to use message sequence graphs (MSGs). A message sequence graph allows the protocol designer to write a finite specification which combines MSCs using basic operations such as branching choice, composition and iteration. The MSC languages described by MSGs are not necessarily regular in the sense of [HM+99]. We characterize here the class of regular MSC languages that are MSG-definable in terms of a notion called finitely generated MSC languages. We show that a regular MSC language is MSG-definable if and only if it is finitely generated. In fact we show that the subclass of "bounded" MSGs defined in [AY99] exactly capture the class of finitely generated regular MSC languages.

1 Introduction

Message sequence charts (MSCs) are an appealing visual formalism often used to capture system requirements in the early design stages. They are particularly suited for describing scenarios for distributed telecommunication software [RGG96,ITU97]. They also appear in the literature as timing sequence diagrams, message flow diagrams and object interaction diagrams and are used in a number of software engineering methodologies [BJR97,HG97,RGG96]. In its basic form, an MSC depicts a single partially-ordered execution of a distributed system which just describes the exchange of messages between the processes of the system. A collection of MSCs is used to capture the scenarios that a designer might want the system to exhibit (or avoid).

Message Sequence Graphs (MSGs) are a nice mechanism for defining collections of MSCs. An MSG is a finite directed graph with a designated initial vertex and terminal vertex in which each node is labelled by an MSC and the edges represent a natural concatenation operation on MSCs. The collection of MSCs defined by an MSG consists of all those MSCs obtained by tracing a path in the MSG from the initial vertex to the terminal vertex and concatenating the MSCs

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 675–686, 2000.
© Springer-Verlag Berlin Heidelberg 2000

that are encountered along the path. It is easy to see that this way of defining a collection of MSCs extends smoothly to the case where there are multiple terminal nodes. Throughout what follows we shall assume this extended notion of an MSG (that is, with multiple terminal nodes). For ease of presentation, we shall also not deal with the so called hierarchical MSGs [AY99].

Intuitively, not every MSG-definable collection of MSCs can be realized as a finite-state device. To formalize this idea we have introduced a notion of a *regular* collection of MSCs and studied its basic properties [HM+99]. Our notion of regularity is independent of the notion of MSGs.

Our main goal in this paper is to pin down the regular MSC languages that can be defined using MSGs. We introduce the notion of an MSC language being *finitely generated*. From our results, which we detail below, it follows that a regular MSC language is MSG-definable if and only if it is finitely generated. In fact we establish the following results.

As already mentioned, not every MSG defines a regular MSC language. Alur and Yannakakis have identified a syntactic property called *boundedness* and shown that the set of all linearizations of the MSCs defined by a bounded MSG is a regular string language over an appropriate alphabet of events. It then follows easily that, in the present setting, every bounded MSG defines a finitely generated regular MSC language. One of our main results here is that the converse is also true, namely, every finitely generated regular MSC language can be defined by a bounded MSG. Since every MSG (bounded or otherwise) defines only a finitely generated MSC language, it follows that a regular MSC language is finitely generated if and only if it is MSG-definable and, in fact, if and only if it is bounded MSG-definable.

Since the class of regular MSC languages strictly includes the class of finitely generated regular MSC languages, one could ask when a regular MSC language is finitely generated. We show that this question is decidable. Finally, one can also ask whether a given MSG defines a regular MSC language (and is hence "equivalent" to a bounded MSG). We show that this decision problem is undecidable.

Turning briefly to related literature, a number of studies are available which are concerned with individual MSCs in terms of their semantics and properties [AHP96,LL95]. A variety of algorithms have been developed for MSGs in the literature—for instance, pattern matching [LP97,Mus99,MPS98], detection of process divergence and non-local choice [BL97], and confluence and race conditions [MP99]. A systematic account of the various model-checking problems associated with MSGs and their complexities can be found in [AY99]. Finally, many of our proof techniques make use of results from the theory of Mazurkiewicz traces [DR95].

In the next section we introduce MSCs and regular MSC languages. We then introduce message sequence graphs in Section 3. In Section 4 we define finitely generated MSC languages and provide an effective procedure to decide whether a regular MSC language is finitely generated. Our main result that the class of finitely generated regular MSC languages coincides with the class of bounded

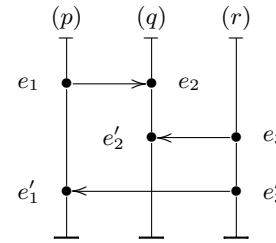


Fig. 1. An example MSC over $\{p, q, r\}$.

MSG-definable languages is then established. Finally, we sketch why the problem of determining if an MSG defines a regular MSC language is undecidable. Due to lack of space, we give here only the main technical constructions and sketches of proofs. All the details are available in [HM+99].

2 Regular MSC Languages

We fix a finite set of processes (or agents) $\mathcal{P}$ and let p, q, r range over $\mathcal{P}$. For each $p \in \mathcal{P}$ we define $\Sigma_p = \{p!q \mid p \neq q\} \cup \{p?q \mid p \neq q\}$ to be the set of communication actions in which p participates. The action $p!q$ is to be read as p *sends to* q and the action $p?q$ is to be read as p *receives from* q. At our level of abstraction, we shall not be concerned with the actual messages that are sent and received. We will also not deal with the internal actions of the agents. We set $\Sigma = \bigcup_{p \in \mathcal{P}} \Sigma_p$ and let a, b range over Σ. We also denote the set of *channels* by $Ch = \{(p, q) \mid p \neq q\}$ and let c, d range over Ch.

A Σ-labelled poset is a structure $M = (E, \leq, \lambda)$ where $(E, \leq)$ is a poset and $\lambda : E \to \Sigma$ is a labelling function. For $e \in E$ we define $\downarrow e = \{e' \mid e' \leq e\}$. For $p \in \mathcal{P}$ and $a \in \Sigma$, we set $E_p = \{e \mid \lambda(e) \in \Sigma_p\}$ and $E_a = \{e \mid \lambda(e) = a\}$, respectively. For each $c \in Ch$, we define the *communication relation* $R_c = \{(e, e') \mid \lambda(e) = p!q, \lambda(e') = q?p$ and $|\downarrow e \cap E_{p!q}| = |\downarrow e' \cap E_{q?p}|\}$. Finally, for each $p \in \mathcal{P}$, we define the *p-causality relation* $R_p = (E_p \times E_p) \cap \leq$.

An MSC (over $\mathcal{P}$) is a *finite* Σ-labelled poset $M = (E, \leq, \lambda)$ which satisfies the following conditions:

(i) Each R_p is a linear order.
(ii) If $p \neq q$ then $|E_{p!q}| = |E_{q?p}|$.
(iii) $\leq = (R_{\mathcal{P}} \cup R_{Ch})^*$ where $R_{\mathcal{P}} = \bigcup_{p \in \mathcal{P}} R_p$ and $R_{Ch} = \bigcup_{c \in Ch} R_c$.

In diagrams, the events of an MSC are presented in *visual order*. The events of each process are arranged in a vertical line and the members of the relation R_{Ch} are displayed as horizontal or downward-sloping directed edges. We illustrate the idea with an example, shown in Figure 1.

Here $\mathcal{P} = \{p, q, r\}$. For $x \in \mathcal{P}$, the events in E_x are arranged along the line labelled (x) with earlier (relative to $\leq$) events appearing above the later

events. The R_{Ch}-edges across agents are depicted by horizontal edges—for instance $e_3 \; R_{(r,q)} \; e_2'$. The labelling function λ is easy to extract from the diagram—for example, $\lambda(e_3') = r!p$ and $\lambda(e_2) = q?p$.

We define regular MSC languages in terms of their linearizations. For the MSC $M = (E, \leq, \lambda)$, let $Lin(M) = \{\lambda(\pi) \mid \pi \text{ is a linearization of } (E, \leq)\}$. By abuse of notation, we have used λ to also denote the natural extension of λ to E^*. The string $p!q \; r!q \; q?p \; q?r \; r!p \; p?r$ is a linearization of the MSC in Figure 1.

We say that $\sigma \in \Sigma^*$ is *proper* if for every prefix τ of σ and every pair (p, q) of processes, $|\tau|_{p!q} \geq |\tau|_{q?p}$. We say that σ is *complete* if σ is proper and $|\sigma|_{p!q} = |\sigma|_{q?p}$ for every pair of processes (p, q). Clearly, any linearization of an MSC is a complete. Conversely, every complete sequence is the linearization of some MSC.

Henceforth, we identify an MSC with its isomorphism class. We let $\mathcal{M}_{\mathcal{P}}$ be the set of MSCs over $\mathcal{P}$. An MSC language $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P}}$ is said to *regular* if $\bigcup\{Lin(M) \mid M \in \mathcal{L}\}$ is a regular subset of Σ^*. We note that the entire set $\mathcal{M}_{\mathcal{P}}$ is not regular by this definition.

We define $L \subseteq \Sigma^*$ to be a *regular string MSC language* if there exists a regular MSC language $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P}}$ such that $L = \bigcup\{Lin(M) \mid M \in \mathcal{L}\}$. As shown in [HM+99], regular MSC languages and regular string MSC languages represent each other. Hence, abusing terminology, we will write "regular MSC language" to mean "regular string MSC language". From the context, it should be clear whether we are working with MSCs from $\mathcal{M}_{\mathcal{P}}$ or complete words over Σ^*.

Given a regular subset $L \subseteq \Sigma^*$, we can decide whether L is a regular MSC language. We say that a state s in a finite-state automaton is *live* if there is a path from s to a final state. Let $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$ be the minimal DFA representing L. Then it is not difficult to see that L is a regular MSC language if and only if we can associate with each live state $s \in S$, a (unique) channel-capacity function $\mathcal{K}_s : Ch \to \mathbb{N}$ which satisfies the following conditions.

(i) If $s \in \{s_{in}\} \cup F$ then $\mathcal{K}_s(c) = 0$ for every $c \in Ch$.
(ii) If s, s' are live states and $\delta(s, p!q) = s'$ then $\mathcal{K}_{s'}((p, q)) = \mathcal{K}_s((p, q))+1$ and $\mathcal{K}_{s'}(c) = \mathcal{K}_s(c)$ for every $c \neq (p, q)$.
(iii) If s, s' are live states and $\delta(s, q?p) = s'$ then $\mathcal{K}_s((p, q)) > 0$, $\mathcal{K}_{s'}((p, q)) = \mathcal{K}_s((p, q))-1$ and $\mathcal{K}_{s'}(c) = \mathcal{K}_s(c)$ for every $c \neq (p, q)$.
(iv) Suppose $\delta(s, a) = s_1$ and $\delta(s_1, b) = s_2$ with $a \in \Sigma_p$ and $b \in \Sigma_q$, $p \neq q$. If $(a, b) \notin Com$ or $\mathcal{K}_s((p, q)) > 0$, there exists s_1' such that $\delta(s, b) = s_1'$ and $\delta(s_1', a) = s_2$. (Here and elsewhere $Com = \{(p!q, q?p) \mid p \neq q\}$.)

In the minimal DFA $\mathcal{A}$ representing a regular MSC language, if s is a live state and $a, b \in \Sigma$ then we say that a and b are *independent at* s if $(a, b) \in Com$ implies $\mathcal{K}_s((p, q)) > 0$ where $\mathcal{K}$ is the unique channel-capacity function associated with $\mathcal{A}$ and $a = p!q$ and $b = q?p$.

We conclude this section by introducing the notion of B-bounded MSC languages. Let $B \in \mathbb{N}$ be a natural number. We say that a word σ in Σ^* is B-bounded if for each prefix τ of σ and for each channel $(p, q) \in Ch$, $|\tau|_{p!q} - |\tau|_{q?p} \leq B$. We say that $L \subseteq \Sigma^*$ is B-bounded if every word $\sigma \in L$ is B-bounded. It is not difficult to show:

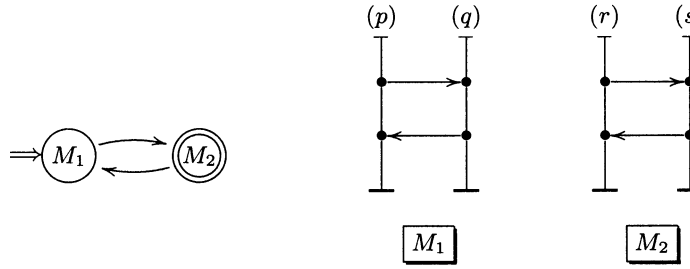


Fig. 2. An example MSG.

Proposition 2.1. *Let L be a regular MSC language. There is a bound $B \in \mathbb{N}$ such that L is B-bounded.*

3 Message Sequence Graphs

An MSG allows a protocol designer to write in a standard way [ITU97], a finite specification which combines MSCs using operations such as branching choice, composition and iteration. Each node is labelled by an MSC and the edges represent the natural operation of MSC concatenation.

To bring out this concatenation operation, we let $M_1 = (E_1, \leq_1, \lambda_1)$ and $M_2 = (E_2, \leq_2, \lambda_2)$ be a pair for MSCs such that E_1 and E_2 are disjoint. For $i \in \{1, 2\}$, let R_c^i and $\{R_p^i\}_{p \in \mathcal{P}}$ denote the underlying communication and process causality relations in M_i. The *(asynchronous) concatenation* of M_1 and M_2, denoted $M_1 \circ M_2$, is the MSC $(E, \leq, \lambda)$ where $E = E_1 \cup E_2$, $\lambda(e) = \lambda_i(e)$ if $e \in E_i$, $i \in \{1, 2\}$, and $\leq = (R_{\mathcal{P}} \cup R_{Ch})^*$, where $R_p = R_p^1 \cup R_p^2 \cup \{(e_1, e_2) \mid e_1 \in E_1, e_2 \in E_2, \lambda(e_1) \in \Sigma_p, \lambda(e_2) \in \Sigma_p\}$ for $p \in \mathcal{P}$, and $R_c = R_c^1 \cup R_c^2$ for $c \in Com$.

A *Message Sequence Graph (MSG)* is a structure $\mathcal{G} = (Q, \rightarrow, Q_{in}, F, \Phi)$, where:

- Q is a finite and nonempty set of states.
- $\rightarrow \subseteq Q \times Q$.
- $Q_{in} \subseteq Q$ is a set of initial states.
- $F \subseteq Q$ is a set of final states.
- $\Phi : Q \rightarrow \mathcal{M}_{\mathcal{P}}$ is a (state-)labelling function.

A *path* π through an MSG $\mathcal{G}$ is a sequence $q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ such that $(q_{i-1}, q_i) \in \rightarrow$ for $i \in \{1, 2, \ldots, n\}$. The MSC generated by π is $M(\pi) = M_0 \circ M_1 \circ M_2 \circ \cdots \circ M_n$, where $M_i = \Phi(q_i)$. A path $\pi = q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ is a *run* if $q_0 \in Q_{in}$ and $q_n \in F$. The language of MSCs accepted by $\mathcal{G}$ is $\mathcal{L}(\mathcal{G}) = \{M(\pi) \in \mathcal{M}_{\mathcal{P}} \mid \pi \text{ is a run through } \mathcal{G}\}$.

An example of an MSG is depicted in Figure 2. It's not hard to see that the MSC language $\mathcal{L}$ defined is *not* regular in the sense defined in Section 2. To see this, we note that $\mathcal{L}$ projected to $\{p!q, r!s\}$ is not a regular string language.

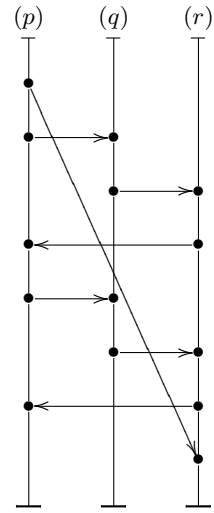


Fig. 3. The atomic MSC M_2 of the non-finitely-generated language $\mathcal{L}_{inf}$.

4 Bounded MSGs and Finitely Generated MSC Languages

Let $\mathcal{X}_1, \mathcal{X}_2 \subseteq \mathcal{M}_{\mathcal{P}}$ be two sets of MSCs. As usual, $\mathcal{X}_1 \circ \mathcal{X}_2$ denotes the pointwise concatenation of $\mathcal{X}_1$ and $\mathcal{X}_2$ given by $\{M \mid \exists M_1 \in \mathcal{X}_1, M_2 \in \mathcal{X}_2 : M = M_1 \circ M_2\}$. For $\mathcal{X} \subseteq \mathcal{M}_{\mathcal{P}}$, we define $\mathcal{X}^0 = \{\varepsilon\}$, where ε denotes the empty MSC, and for $i \geq 0$, $\mathcal{X}^{i+1} = \mathcal{X} \circ \mathcal{X}^i$. The *asynchronous iteration of* $\mathcal{X}$ is then defined by $\mathcal{X}^{\circledast} = \bigcup_{i \geq 0} \mathcal{X}^i$.

Let $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P}}$. We say that $\mathcal{L}$ is *finitely generated* if there is a finite set of MSCs $\mathcal{X} \subseteq \mathcal{M}_{\mathcal{P}}$ such that $\mathcal{L} \subseteq \mathcal{X}^{\circledast}$.

We first observe that not every regular MSC language is finitely generated. Let $\mathcal{P} = \{p, q, r\}$. Consider the regular expression $p!q\ \sigma^*\ q?p$, where σ is the sequence $p!r\ r?p\ r!q\ q?r\ q!p\ p?q$. This expression describes an infinite set of MSCs $\mathcal{L}_{inf} = \{M_i\}_{i=0}^{\omega}$. Figure 3 shows the MSC M_2. None of the MSCs in this language can be expressed as the concatenation of two or more non-trivial MSCs. Hence, this language is *not* finitely generated.

Our interest in finitely generated languages stems from the fact that MSGs can define only finitely generated MSC languages. We now wish to decide whether a regular MSC language is finitely generated. To do this we shall make use of the notion of atoms.

Let $M, M' \in \mathcal{M}_{\mathcal{P}}$ be nonempty MSCs. Then M' *is a component of* M in case there exist $M_1, M_2 \in \mathcal{M}$ such that $M = M_1 \circ M' \circ M_2$. We say that M *is an atom* if the only component of M is M itself. Thus, an atom is a nonempty message sequence chart that cannot be decomposed into non-trivial subcomponents. For an MSC M we let $Comp(M)$ denote the set $\{M' \mid M' \text{ is a component of } M\}$. We let $Atoms(M)$ denote the set $\{M' \mid M' \text{ is an atom and } M' \text{ is a component of } M\}$.

For an MSC language $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P}}$, $Comp(\mathcal{L}) = \bigcup\{Comp(M) \mid M \in \mathcal{L}\}$ and $Atoms(\mathcal{L}) = \bigcup\{Atoms(M) \mid M \in \mathcal{L}\}$. It is clear that the question of deciding whether L is finitely generated is equivalent to checking whether $Atoms(\mathcal{L})$ is finite.

Theorem 4.1. *Let $\mathcal{L}$ be a regular MSC language. It is decidable whether $\mathcal{L}$ is finitely generated.*

Proof Sketch: Let $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$ be the minimum DFA for $\mathcal{L}$. From $\mathcal{A}$, we construct a finite family of finite-state automata which together accept the linearizations of the MSCs in $Atoms(\mathcal{L})$. It will then follow that $\mathcal{L}$ is finitely generated if and only if each of these automata accepts a finite language. We sketch the details below.

We know that for each live state $s \in S$, we can assign a capacity function $\mathcal{K}_s : Ch \to \mathbb{N}$ which counts the number of messages present in each channel when the state s is reached. We say that s is a *zero-capacity state* if $\mathcal{K}_s(c) = 0$ for each channel c. The following claims are easy to prove.

Claim 1: Let M be an MSC in $Comp(\mathcal{L})$ (in particular, in $Atoms(\mathcal{L})$) and w be a linearization of M. Then, there are zero-capacity live states s, s' in $\mathcal{A}$ such that $s \xrightarrow{w} s'$.

Claim 2: Let M be an MSC in $Comp(\mathcal{L})$. M is an atom if and only if for each linearization w of M and each pair (s, s') of zero-capacity live states in $\mathcal{A}$, if $s \xrightarrow{w} s'$ then no intermediate state visited in this run has zero-capacity.

Let us say that two complete words w and w' are equivalent, written $w \sim w'$, if they are linearizations of the same MSC. Suppose $s \xrightarrow{w} s'$ and $w \sim w'$. Then it is easy to see that $s \xrightarrow{w'} s'$ as well.

With each pair (s, s') of live zero-capacity states we associate a language $L_{At}(s, s')$. A word w belongs to $L_{At}(s, s')$ if and only if w is complete, $s \xrightarrow{w} s'$ and for each $w' \sim w$ the run $s \xrightarrow{w'} s'$ has no zero-capacity intermediate states. From the two claims proved above, each of these languages consists of all the linearizations of some subset of $Atoms(\mathcal{L})$ and the linearizations of each element of $Atoms(\mathcal{L})$ is contained in some $L_{At}(s, s')$. Thus, it suffices to check for the finiteness of each of these languages.

Let $L_{s,s'}$ be the language of strings accepted by $\mathcal{A}$ when we set the initial state to be s and the set of final states to be $\{s'\}$. Clearly $L_{At}(s, s') \subseteq L_{s,s'}$. We now show how to construct an automaton for for $L_{At}(s, s')$.

We begin with $\mathcal{A}$ and prune the automaton as follows:

- Remove all incoming edges at s and all outgoing edges at s'.
- If $t \notin \{s, s'\}$ and $\mathcal{K}_t = \overline{0}$, remove t and all its incoming and outgoing edges.
- Recursively remove all states that become unreachable as a result of the preceding operation.

Let $\mathcal{B}$ be the resulting automaton. B accepts any complete word w on which the run from s to s' does not visit an intermediate zero-capacity state. Clearly,

$L_{At}(s, s') \subseteq L(\mathcal{B})$. However, $L(\mathcal{B})$ may also contain linearizations of non-atomic MSCs that happen to have no complete prefix. For all such words, we know from Claim 2 that there is at least one equivalent linearization on which the run passes through a zero-capacity state and which would hence be eliminated from $L(\mathcal{B})$. Thus, $L_{At}(s, s')$ is the $\sim$-closed subset of $L(\mathcal{B})$ and we need to prune $\mathcal{B}$ further to obtain the automaton for $L_{At}(s, s')$.

Recall that the original DFA $\mathcal{A}$ was structurally closed with respect to the independence relation on communication actions in the following sense. Suppose $\delta(s_1, a) = s_2$ and $\delta(s_2, b) = s_3$ with a, b independent at s_1. Then, there exists s_2' such that $\delta(s_1, b) = s_2'$ and $\delta(s_2', a) = s_3$.

To identify the closed subset of $L(\mathcal{B})$, we look for local violations of this "diamond" property and carefully prune transitions. We first blow up the state space into triples of the form (s_1, s_2, s_3) such that there exist a and a' with $\delta(s_1, a) = s_2$ and $\delta(s_2, a') = s_3$. Let S' denote this set of triples. We obtain a nondeterministic transition relation $\delta' = \{((s_1, s_2, s_3), a, (t_1, t_2, t_3)) \mid s_2 = t_1, s_3 = t_2, \delta(s_2, a) = s_3\}$. Set $S_{in} = \{(s_1, s_2, s_3) \in S' \mid s_2 = s_{in}\}$ and $F' = \{(s_1, s_f, s_2) \in S' \mid s_f \in F\}$. Let $\mathcal{B}' = (S', \Sigma, \delta', S_{in}, F')$.

Consider any state s_1 in $\mathcal{B}$ such that a and b are independent at s_1, $\delta(s_1, a) = s_2$, $\delta(s_2, b) = s_3$ but there is no s_2' such that $\delta(s_1, b) = s_2'$ and $\delta(s_2', a) = s_3$. For each such s_1, we remove all transitions of the form $((t, s_0, s_1), a, (s_0, s_1, t'))$ and $((t, s_2, s_3), b, (s_2, s_3, t'))$ from $\mathcal{B}'$. We then recursively remove all states which become unreachable after this pruning.

Eventually, we arrive at an automaton $\mathcal{C}$ such that $L(\mathcal{C}) = L_{At}(s, s')$. Since $\mathcal{C}$ is a finite-state automaton, we can easily check whether $L(\mathcal{C})$ is finite. This process is repeated for each pair of live zero-capacity states. □

Alur and Yannakakis [AY99] introduced the notion of boundedness for MSGs. They also showed that the set of all linearizations of the MSCs defined by a bounded MSG is a regular string language. In the present setting this boils down to boundedness of an MSG being a sufficient condition for its MSC language to be regular. To state their condition, we first have to define the notion of connectedness.

Let $M = (E, \leq, \lambda)$ be an MSC. We let CG_M, *the communication graph of* M, be the directed graph $(\mathcal{P}, \mapsto)$ defined as follows: $(p, q) \in \mapsto$ if and only if there exists an $e \in E$ with $\lambda(e) = p!q$. M is *connected* if CG_M consists of one non-trivial strongly connected component and isolated vertices. For an MSC language $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P}}$ we say that $\mathcal{L}$ *is connected* in case every $M \in \mathcal{L}$ is connected.

Let $\mathcal{G} = (Q, \rightarrow, Q_{in}, F, \Phi)$ be an MSG. A *loop* in $\mathcal{G}$ is a sequence of edges that starts and ends at the same node. We say that $\mathcal{G}$ is *bounded* if for every loop $\pi = q \rightarrow q_1 \rightarrow \cdots \rightarrow q$, the MSC $M(\pi)$ is connected. An MSC language $\mathcal{L}$ is a *bounded MSG-language* if there exists a bounded MSG $\mathcal{G}$ with $\mathcal{L} = \mathcal{L}(\mathcal{G})$.

It is easy to check whether a given MSG is bounded. Clearly, the MSG of Figure 2 is *not* bounded. One of the main results concerning bounded MSGs shown in [AY99] at once implies:

Lemma 4.2. *Every bounded MSG-language is a regular MSC language.*

Our main interest in this section is to prove the converse of Lemma 4.2.

Lemma 4.3. *Let $\mathcal{L}$ be a finitely generated regular MSC language. Then, $\mathcal{L}$ is a bounded MSG-language.*

Proof Sketch: Suppose $\mathcal{L}$ is a regular MSC language accepted by the minimal DFA $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$. Let $Atoms(\mathcal{L}) = \{a_1, a_2, \ldots, a_m\}$. For each atom a_i, fix a linearization $u_i \in Lin(a_i)$. Define an auxiliary DFA $\mathcal{B} = (S^{\overline{0}}, Atoms(\mathcal{L}), s_{in}, \widehat{\delta}, \widehat{F})$ as follows:

- $S^{\overline{0}}$ is the set of states of $\mathcal{A}$ which have zero-capacity functions.
- $\widehat{F} = F$
- $\widehat{\delta}(s, a_i) = s'$ iff $\delta(s, u_i) = s'$ in $\mathcal{A}$. (Note that $u, u' \in Lin(a_i)$ implies $\delta(s, u) = \delta(s, u')$, so s' is fixed independent of the choice of $u_i \in Lin(a_i)$.)

Thus, $\mathcal{B}$ accepts the (regular) language of atoms corresponding to $\mathcal{L}(\mathcal{A})$. We can define a natural independence relation I_A on atoms as follows: atoms a_i and a_j are independent if and only if the set of active processes in a_i is disjoint from the set of active processes in a_j. (The process p is *active* in the MSC $(E, \leq, \lambda)$ if E_p is non-empty.)

It follows that $L(\mathcal{B})$ is a regular Mazurkiewicz trace language over the trace alphabet $(Atoms(\mathcal{L}), I_A)$. As usual, for $w \in Atoms(\mathcal{L})^*$, we let $[w]$ denote the equivalence class of w with respect to I_A.

We now fix a strict linear order $\prec$ on $Atoms(\mathcal{L})$. This induces a (lexicographic) total order on words over $Atoms(\mathcal{L})$. Let $\textsc{Lex} \subseteq Atoms(\mathcal{L})^*$ be given by: $w \in \textsc{Lex}$ iff w is the lexicographically least element in $[w]$.

For a trace language L over $(Atoms(\mathcal{L}), I_A)$, let $lex(L)$ denote the set $L \cap \textsc{Lex}$.

Remark 4.4 ([DR95], Sect. 6.3.1)

(i) If L is a regular trace language over $(Atoms(\mathcal{L}), I_A)$, then $lex(L)$ is a regular language over $Atoms(\mathcal{L})$. Moreover, $L = \{[w] \mid w \in lex(L))$.
(ii) If $w_1 w w_2 \in \textsc{Lex}$, then $w \in \textsc{Lex}$.
(iii) If w is not *a connected*[1] *trace, then $ww \notin \textsc{Lex}$.*

¿From (i) we know that $lex(L(\mathcal{B}))$ is a regular language over $Atoms(\mathcal{L})$. Let $\mathcal{C} = (S', Atoms(\mathcal{L}), s'_{in}, \delta', F')$ be the DFA over $Atoms(\mathcal{L})$ obtained by eliminating the (unique) dead state, if any, from the minimal DFA for $lex(L(\mathcal{B}))$. It is easy to see that an MSC M belongs to $\mathcal{L}$ if and only if it can be decomposed into a sequence of atoms accepted by $\mathcal{C}$. Using this fact, we can derive an MSG $\mathcal{G}$ from $\mathcal{C}$ such that $\mathcal{L}(\mathcal{G}) = \mathcal{L}$. We define $\mathcal{G} = (Q, \rightarrow, Q_{in}, F, \Phi)$ as follows:

- $Q = S' \times (Atoms(\mathcal{L}) \cup \{\varepsilon\})$.

[1] A trace is said to be *connected* if, when viewed as a labelled partial order, its Hasse diagram consists of a single connected component. See [DR95] for a more formal definition.

- $Q_{in} = \{(s'_{in}, \varepsilon)\}$.
- $(s,b) \to (s',b')$ iff $\delta'(s,b') = s'$.
- $F' = F \times Atoms(\mathcal{L})$.
- $\Phi(s,b) = b$.

Clearly $\mathcal{G}$ is an MSG and the MSC language that it defines is $\mathcal{L}$. We need to show that $\mathcal{G}$ is bounded. To this end, let $\pi = (s,b) \to (s_1,b_1) \to \cdots \to (s_n,b_n) \to (s,b)$ be a loop in $\mathcal{G}$. We need to establish that the MSC $M(\pi) = b_1 \circ \cdots \circ b_n \circ b$ defined by this loop is connected. Let $w = b_1 b_2 \ldots b_n b$.

Consider the corresponding loop $s \xrightarrow{b_1} s_1 \xrightarrow{b_2} \cdots \xrightarrow{b_n} s_n \xrightarrow{b} s$ in $\mathcal{C}$. Since every state in $\mathcal{C}$ is live, there must be words w_1, w_2 over $Atoms(\mathcal{L})$ such that $w_1 w^k w_2 \in lex(L(\mathcal{B}))$ for every $k \geq 0$.

¿From (ii) of Remark 4.4, $w^k \in$ Lex. This means, by (iii) of Remark 4.4, that w describes a connected trace over $(Atoms(\mathcal{L}), I_A)$. From this, it is not difficult to see that the underlying undirected graph of the communication graph $CG_{M(\pi)} = (\mathcal{P}, \mapsto)$ consists of a single connected component $C \subseteq \mathcal{P}$ and isolated processes. We have to argue that the component C is, in fact, *strongly* connected. We show that if C is not strongly connected, then the regular MSC language $\mathcal{L}$ is not B-bounded for any $B \in \mathbb{N}$, thus contradicting Proposition 2.1.

Suppose that the underlying graph of C is connected but C not strongly connected. Then, there exist two processes $p, q \in C$ such that $p \mapsto q$, but there is no path from q back to p in $CG_{M(\pi)}$. For $k \geq 0$, let $M(\pi)^k = (E, \leq, \lambda)$ be the MSC corresponding to the k-fold iteration $\underbrace{M(\pi) \circ M(\pi) \circ \cdots \circ M(\pi)}_{k \text{ times}}$. Since $p \mapsto q$ in $CG_{M(\pi)}$, it follows that there are events labelled $p!q$ and $q?p$ in $M(\pi)$. Moreover, since there is no path from q back to p in $CG_{M(\pi)}$, we can conclude that in $M(\pi)^k$, for each event e with $\lambda(e) = p!q$, there is no event labelled $q?p$ in $\downarrow e$. This means that $M(\pi)^k$ admits a linearization v'_k with a prefix τ'_k which includes all the events labelled $p!q$ and excludes all the events labelled $q?p$, so that $|\tau|_{p!q} - |\tau|_{q?p} \geq k$.

By Proposition 2.1, since $\mathcal{L}$ is a regular MSC language, there is be a bound $B \in \mathbb{N}$ such that every word in $\mathcal{L}$ is B-bounded—that is, for each $v \in \mathcal{L}$, for each prefix τ of v and for each channel $(p,q) \in Ch$, $|\tau|_{p!q} - |\tau|_{q?p} \leq B$. Recall that $w_1 w^k w_2 \in lex(L(\mathcal{B}))$ for every $k \geq 0$. Fix linearizations v_1 and v_2 of the atom sequences w_1 and w_2, respectively. Then, for every $k \geq 0$, $u_k = v_1 v'_k v_2 \in \mathcal{L}$ where v'_k is the linearization of $M(\pi)^k$ defined earlier. Setting k to be $B+1$, we find that u_k admits a prefix $\tau_k = v_1 \tau'_k$ such that $|\tau_k|_{p!q} - |\tau_k|_{q?p} \geq B+1$, which contradicts the B-boundedness of $\mathcal{L}$.

Hence, it must be the case that C is a strongly connected component, which establishes that the MSG $\mathcal{G}$ we have constructed is bounded. □

Putting together Lemmas 4.2 and 4.3, we obtain the following characterization of MSG-definable regular MSC languages.

Theorem 4.5. *Let $\mathcal{L}$ be a regular MSC language. Then the following statements are equivalent:*

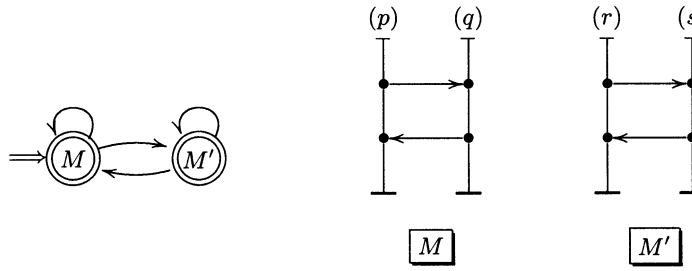


Fig. 4. An non-bounded MSG whose language is regular.

(i) $\mathcal{L}$ *is finitely generated.*
(ii) $\mathcal{L}$ *is a bounded MSG-language.*
(iii) $\mathcal{L}$ *is MSG-definable.*

It is easy to see that boundedness is not a necessary condition for regularity. Consider the MSG in Figure 4, which is not bounded. It accepts the regular MSC language $M \circ (M + M')^{\circledast}$.

Thus, it would be useful to provide a characterization of the class of MSGs representing regular MSC languages. Unfortunately, the following result shows that there is no (recursive) characterization of this class.

Theorem 4.6. *The problem of deciding whether a given MSG represents a regular MSC language is undecidable.*

Proof Sketch: It is known that the problem of determining whether the trace-closure of a regular language $L \subseteq A^*$ with respect to a trace alphabet (A, I) is also regular is undecidable [Sak92]. We reduce this problem to the problem of checking whether the MSC language defined by an MSG is regular.

Let (A, I) be a trace alphabet. We fix a set of processes $\mathcal{P}$ and the associated communication alphabet Σ and encode each letter a by an MSC M_a over $\mathcal{P}$ such that the communication graph CG_{M_a} is strongly connected. Moreover, if $(a, b) \in I$, then the sets of active processes of M_a and M_b are disjoint. The encoding ensures that we can construct a finite-state automaton to parse any word over Σ and determine whether it arises as the linearization of an MSC of the form $M_{a_1} \circ M_{a_2} \circ \cdots \circ M_{a_k}$. If so, the parser can uniquely reconstruct the corresponding word $a_1 a_2 \ldots a_k$ over A.

Let $\mathcal{A}$ be the minimal DFA corresponding to a regular language L over A. We construct an MSG $\mathcal{G}$ from $\mathcal{A}$ as described in the proof of Lemma 4.3. Given the properties of our encoding, we can then establish that the MSC language $L(\mathcal{G})$ is regular if and only if the trace-closure of L is regular, thus completing the reduction. □

References

AHP96. Alur, R., Holzmann, G.J., and Peled, D.: An analyzer for message sequence charts. *Software Concepts and Tools*, **17**(2) (1996) 70–77.

AY99. Alur, R., and Yannakakis, M.: Model checking of message sequence charts. *Proc. CONCUR'99*, LNCS **1664**, Springer Verlag (1999) 114–129.

BL97. Ben-Abdallah, H., and Leue, S.: Syntactic detection of process divergence and non-local choice in message sequence charts. *Proc. TACAS'97*, LNCS **1217**, Springer-Verlag (1997) 259–274.

BJR97. Booch, G., Jacobson, I., and Rumbaugh, J.: *Unified Modeling Language User Guide.* Addison Wesley (1997).

DR95. Diekert, V., and Rozenberg, G. (Eds.): *The book of traces.* World Scientific (1995).

HG97. Harel, D., and Gery, E.: Executable object modeling with statecharts. *IEEE Computer*, July 1997 (1997) 31–42.

HM+99. Henriksen, J.G., Mukund, M., Narayan Kumar, K., and Thiagarajan, P.S.: Towards a theory of regular MSC languages. BRICS Report RS-99-52, Department of Computer Science, Aarhus University, Denmark (1999).

ITU97. ITU-TS Recommendation Z.120: *Message Sequence Chart (MSC).* ITU-TS, Geneva (1997)

LL95. Ladkin, P.B., and Leue, S.: Interpreting message flow graphs. *Formal Aspects of Computing* **7**(5) (1975) 473–509.

LP97. Levin, V., and Peled, D.: Verification of message sequence charts via template matching. *Proc. TAPSOFT'97*, LNCS **1214**, Springer-Verlag (1997) 652–666.

Mus99. Muscholl, A.: Matching specifications for message sequence charts. *Proc. FOSSACS'99*, LNCS **1578**, Springer-Verlag (1999) 273–287.

MP99. Muscholl, A., and Peled, D.: Message sequence graphs and decision problems on Mazurkiewicz traces. *Proc. MFCS'99*, LNCS **1672**, Springer-Verlag (1999) 81–91.

MPS98. Muscholl, A., Peled, D., and Su, Z.: Deciding properties for message sequence charts. *Proc. FOSSACS'98*, LNCS **1378**, Springer-Verlag (1998) 226–242.

RGG96. Rudolph, E., Graubmann, P., and Grabowski, J.: Tutorial on message sequence charts. In *Computer Networks and ISDN Systems—SDL and MSC*, Volume 28 (1996).

Sak92. Sakarovitch, J.: The "last" decision problem for rational trace languages. *Proc. LATIN'92*, LNCS **583**, Springer-Verlag (1992) 460–473.

Pseudorandomness

Oded Goldreich

Department of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot, ISRAEL. oded@wisdom.weizmann.ac.il.

Abstract. We postulate that a distribution is pseudorandom if it cannot be told apart from the uniform distribution by any efficient procedure. This yields a robust definition of pseudorandom generators as efficient deterministic programs stretching short random seeds into longer pseudorandom sequences. Thus, pseudorandom generators can be used to reduce the randomness-complexity in any efficient procedure. Pseudorandom generators and computational difficulty are closely related: loosely speaking, each can be efficiently transformed into the other.

1 Introduction

The second half of this century has witnessed the development of three theories of randomness, a notion which has been puzzling thinkers for ages. The first theory (cf. [9]), initiated by Shannon [33], is rooted in probability theory and is focused at distributions that are not perfectly random. Shannon's Information Theory characterizes perfect randomness as the extreme case in which the *information content* is maximized (and there is no redundancy at all).[1] Thus, perfect randomness is associated with a unique distribution – the uniform one. In particular, by definition, one cannot generate such perfect random strings from shorter random strings.

The second theory (cf. [22, 23]), due to Solomonov [34], Kolmogorov [21] and Chaitin [6], is rooted in computability theory and specifically in the notion of a universal language (equiv., universal machine or computing device). It measures the complexity of objects in terms of the shortest program (for a fixed universal machine) that generates the object.[2] Like Shannon's theory, Kolmogorov Complexity is quantitative and perfect random objects appear as an extreme case. Interestingly, in this approach one may say that a single object, rather than a distribution over objects, is perfectly random. Still, Kolmogorov's approach is inherently intractable (i.e., Kolmogorov Complexity is

[1] In general, the amount of information in a distribution D is defined as $-\sum_x D(x)\log_2 D(x)$. Thus, the uniform distribution over strings of length n has information measure n, and any other distribution over n-bit strings has lower information measure. Also, for any function $f:\{0,1\}^n \to \{0,1\}^m$ with $n < m$, the distribution obtained by applying f to a uniformly distributed n-bit string has information measure at most n, which is strictly lower than the length of the output.

[2] For example, the string 1^n has Kolmogorov Complexity $O(1)+\log_2 n$ (by virtue of the program "print n ones" which has length dominated by the binary encoding of n. In contrast, a simple counting argument shows that most n-bit strings have Kolmogorov Complexity at least n (since each program can produce only one string).

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 687–704, 2000.
© Springer-Verlag Berlin Heidelberg 2000

uncomputable), and – by definition – one cannot generate strings of high Kolmogorov Complexity from short random strings.

The third theory, initiated by Blum, Goldwasser, Micali and Yao [17, 4, 37], is rooted in complexity theory and is the focus of this survey. This approach is explicitly aimed at providing a notion of perfect randomness that nevertheless allows to efficiently generate perfect random strings from shorter random strings. The heart of this approach is the suggestion to view objects as equal if they cannot be told apart by any efficient procedure. Consequently a distribution that cannot be efficiently distinguished from the uniform distribution will be considered as being random (or rather called pseudorandom). Thus, randomness is not an "inherent" property of objects (or distributions) but is rather relative to an observer (and its computational abilities). To demonstrate this approach, let us consider the following mental experiment.

> Alice and Bob play HEAD OR TAIL in one of the following four ways. In all of them Alice flips a coin high in the air, and Bob is asked to guess its outcome *before* the coin hits the floor. The alternative ways differ by the knowledge Bob has before making his guess. In the first alternative, Bob has to announce his guess before Alice flips the coin. Clearly, in this case Bob wins with probability $1/2$. In the second alternative, Bob has to announce his guess while the coin is spinning in the air. Although the outcome is *determined in principle* by the motion of the coin, Bob does not have accurate information on the motion and thus we believe that also in this case Bob wins with probability $1/2$. The third alternative is similar to the second, except that Bob has at his disposal sophisticated equipment capable of providing accurate *information* on the coin's motion as well as on the environment effecting the outcome. However, Bob cannot process this information in time to improve his guess. In the fourth alternative, Bob's recording equipment is directly connected to a *powerful computer* programmed to solve the motion equations and output a prediction. It is conceivable that in such a case Bob can improve substantially his guess of the outcome of the coin.

We conclude that the randomness of an event is relative to the information and computing resources at our disposal. Thus, a natural concept of pseudorandomness arises – a distribution is pseudorandom if no efficient procedure can distinguish it from the uniform distribution, where efficient procedures are associated with (probabilistic) polynomial-time algorithms.

Orientation Remarks

We consider finite objects, encoded by binary finite sequences called strings. When we talk of distributions we mean discrete probability distributions having a finite support that is a set of strings. Of special interest is the uniform distribution, that for a length parameter n (explicit or implicit in the discussion), assigns each n-bit string $x \in \{0,1\}^n$ equal probability (i.e., probability 2^{-n}). We will colloquially speak of "perfectly random strings" meaning strings selected according to such a uniform distribution.

We associate efficient procedures with probabilistic polynomial-time algorithms. An algorithm is called polynomial-time if there exists a polynomial p so that for any

possible input x, the algorithm runs in time bounded by $p(|x|)$, where $|x|$ denotes the length of the string x. Thus, the running time of such algorithm grows moderately as a function of the length of its input. A probabilistic algorithm is one that can take random steps, where, without loss of generality, a random step consists of selecting which of two predetermined steps to take next so that each possible step is taken with probability $1/2$. These choices are called the algorithm's internal coin tosses.

Organization, Acknowledgment, and Further Details

Sections 2 and 3 provide a basic treatment of *pseudorandom generators* (as briefly discussed in the abstract). The rest of this survey goes somewhat beyond: In Section 4 we treat *pseudorandom functions*, and in Section 5 we further discuss the practical and conceptual significance of pseudorandom generators. In Section 6 we discuss alternative notions of pseudorandom generators, viewing them all as special cases of a general paradigm. The survey is based on [11, Chap. 3], and the interested reader is referred to there for further details.

2 The Notion of Pseudorandom Generators

Loosely speaking, a pseudorandom generator is an *efficient* program (or algorithm) that *stretches* short random strings into long *pseudorandom* sequences. The latter sentence emphasizes three fundamental aspects in the notion of a pseudorandom generator:

1. Efficiency: The generator has to be efficient. As we associate efficient computations with polynomial-time ones, we postulate that the generator has to be implementable by a deterministic polynomial-time algorithm.
   This algorithm takes as input a string, called its seed. The seed captures a bounded amount of randomness used by a device that "generates pseudorandom sequences." The formulation views any such device as consisting of a deterministic procedure applied to a random seed.
2. Stretching: The generator is required to stretch its input seed to a longer output sequence. Specifically, it stretches n-bit long seeds into $\ell(n)$-bit long outputs, where $\ell(n) > n$. The function ℓ is called the stretching measure (or stretching function) of the generator.
3. Pseudorandomness: The generator's output has to look random to any efficient observer. That is, any efficient procedure should fail to distinguish the output of a generator (on a random seed) from a truly random sequence of the same length. The formulation of the last sentence refers to a general notion of computational indistinguishability, which is the heart of the entire approach.

2.1 Computational Indistinguishability

Intuitively, two objects are called computationally indistinguishable if no efficient procedure can tell them apart. As usual in complexity theory, an elegant formulation requires asymptotic analysis (or rather a functional treatment of the running time of algorithms

in terms of the length of their input).[3] Thus, the objects in question are infinite sequences of distributions, where each distribution has a finite support. Such a sequence will be called a distribution ensemble. Typically, we consider distribution ensembles of the form $\{D_n\}_{n\in\mathbb{N}}$, where for some function $\ell : \mathbb{N}\to\mathbb{N}$, the support of each D_n is a subset of $\{0,1\}^{\ell(n)}$. Furthermore, typically ℓ will be a positive polynomial. For such D_n, we denote by $e\sim D_n$ the process of selecting e according to distribution D_n. Consequently, for a predicate P, we denote by $\Pr_{e\sim D_n}[P(e)]$ the probability that $P(e)$ holds when e is distributed (or selected) according to D_n.

Definition 1 (Computational Indistinguishability [17, 37]): *Two probability ensembles, $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$, are called* computationally indistinguishable *if for any probabilistic polynomial-time algorithm A, for any positive polynomial p, and for all sufficiently large n's*

$$|\Pr_{x\sim X_n}[A(x)=1] - \Pr_{y\sim Y_n}[A(y)=1]| < \frac{1}{p(n)}$$

The probability is taken over X_n (resp., Y_n) as well as over the coin tosses of algorithm A.

A couple of comments are in place. Firstly, we have allowed algorithm A (called a distinguisher) to be probabilistic. This makes the requirement only stronger, and seems essential to several important aspects of our approach. Secondly, we view events occuring with probability that is upper bounded by the reciprocal of polynomials as negligible. This is well-coupled with our notion of efficiency (i.e., polynomial-time computations): An event that occurs with negligible probability (as a function of a parameter n), will also occur with negligible probability if the experiment is repeated for $\mathrm{poly}(n)$-many times.

We note that computational indistinguishability is a strictly more liberal notion than statistical indistinguishability (cf. [37, 15]). An important case is the one of distributions generated by a pseudorandom generator as defined next.

2.2 Basic Definition and Initial Discussion

We are now ready for the main definition. Recall that a stretching function, $\ell:\mathbb{N}\to\mathbb{N}$, satisfies $\ell(n) > n$ for all n.

Definition 2 (Pseudorandom Generators [4, 37]): *A deterministic polynomial-time algorithm G is called a* pseudorandom generator *if there exists a* stretching function, *$\ell : \mathbb{N}\to\mathbb{N}$, so that the following two probability ensembles, denoted $\{G_n\}_{n\in\mathbb{N}}$ and $\{R_n\}_{n\in\mathbb{N}}$, are computationally indistinguishable*

1. *Distribution G_n is defined as the output of G on a uniformly selected seed in $\{0,1\}^n$.*
2. *Distribution R_n is defined as the uniform distribution on $\{0,1\}^{\ell(n)}$.*

[3] We stress that the asymptotic (or functional) treatment is not essential to this approach. One may develop the entire approach in terms of inputs of fixed lengths and an adequate notion of complexity of algorithms. However, such an alternative treatment is more cumbersome.

That is, letting U_m denote the uniform distribution over $\{0,1\}^m$, we require that for any probabilistic polynomial-time algorithm A, for any positive polynomial p, and for all sufficiently large n's

$$|\Pr_{s\sim U_n}[A(G(s))=1] - \Pr_{r\sim U_{\ell(n)}}[A(r)=1]| < \frac{1}{p(n)}$$

Thus, pseudorandom generators are efficient (i.e., polynomial-time) deterministic programs that expand short randomly selected seeds into longer pseudorandom bit sequences, where the latter are defined as computationally indistinguishable from truly random sequences by efficient (i.e., polynomial-time) algorithms. It follows that any efficient randomized algorithm maintains its performance when its internal coin tosses are substituted by a sequence generated by a pseudorandom generator. That is,

Construction 3 (typical application of pseudorandom generators): *Let A be a probabilistic polynomial-time algorithm, and $\rho(n)$ denote an upper bound on its randomness complexity. Let $A(x,r)$ denote the output of A on input x and coin tosses sequence $r \in \{0,1\}^{\rho(|x|)}$. Let G be a pseudorandom generator with stretching function $\ell : \mathbb{N} \to \mathbb{N}$. Then A_G is a randomized algorithm that on input x, proceeds as follows. It sets $k = k(|x|)$ to be the smallest integer such that $\ell(k) \geq \rho(|x|)$, uniformly selects $s \in \{0,1\}^k$, and outputs $A(x,r)$, where r is the $\rho(|x|)$-bit long prefix of $G(s)$.*

It can be shown that it is infeasible to find long x's on which the *input-output behavior* of A_G is *noticeably different* from the one of A, although A_G may use much fewer coin tosses than A. That is

Proposition 4 *Let A and G be as above. For any algorithm D, let $\Delta_{A,D}(x)$ denote the discrepancy, as judged by D, in the behavior of A and A_G on input x. That is,*

$$\Delta_{A,D}(x) \stackrel{\text{def}}{=} |\Pr_{r\sim U_{\rho(n)}}[D(x,A(x,r))=1] - \Pr_{s\sim U_{k(n)}}[D(x,A_G(x,s))=1]|$$

where the probabilities are taken over the U_m's as well as over the coin tosses of D. Then for every pair of probabilistic polynomial-time algorithms, a finder F *and* a distinguisher D, *every positive polynomial p and all sufficiently long n's*

$$\Pr\left[\Delta_{A,D}(F(1^n)) > \frac{1}{p(n)}\right] < \frac{1}{p(n)}$$

where $|F(1^n)| = n$ and the probability is taken over the coin tosses of F.

The proposition is proven by showing that a triplet (A,F,D) violating the claim can be converted into an algorithm D' that distinguishes the output of G from the uniform distribution, in contradiction to the hypothesis. Analogous arguments are applied whenever one wishes to prove that an efficient randomized process (be it an algorithm as above or a multi-party computation) preserves its behavior when one replaces true randomness by pseudorandomness as defined above. Thus, given pseudorandom generators with large stretching function, *one can considerably reduce the randomness complexity in any efficient application*.

2.3 Amplifying the Stretch Function

Pseudorandom generators as defined above are only required to stretch their input a bit; for example, stretching n-bit long inputs to $(n+1)$-bit long outputs will do. Clearly, generator of such moderate stretch function are of little use in practice. In contrast, we want to have pseudorandom generators with an arbitrary long stretch function. By the efficiency requirement, the stretch function can be at most polynomial. It turns out that pseudorandom generators with the smallest possible stretch function can be used to construct pseudorandom generators with any desirable polynomial stretch function. (Thus, when talking about the existence of pseudorandom generators, we may ignore the stretch function.)

Theorem 5 [14]: *Let G be a pseudorandom generator with stretch function $\ell(n) = n+1$, and ℓ' be any polynomially-bounded stretch function, that is polynomial-time computable. Let $G_1(x)$ denote the $|x|$-bit long prefix of $G(x)$, and $G_2(x)$ denote the last bit of $G(x)$ (i.e., $G(x) = G_1(x)\,G_2(x)$). Then*

$$G'(s) \stackrel{\text{def}}{=} \sigma_1\sigma_2\cdots\sigma_{\ell'(|s|)}\,,$$
$$\text{where } x_0 = s,\ \sigma_i = G_2(x_{i-1}) \text{ and } x_i = G_1(x_{i-1}), \text{ for } i = 1, ..., \ell'(|s|)$$

is a pseudorandom generator with stretch function ℓ'.

Proof Sketch: The theorem is proven using the *hybrid technique* (cf. [10, Sec. 3.2.3]): One considers distributions H_n^i (for $i = 0, ..., \ell(n)$) defined by $U_i^{(1)} P_{\ell(n)-i}(U_n^{(2)})$, where $U_i^{(1)}$ and $U_n^{(2)}$ are independent uniform distributions (over $\{0,1\}^i$ and $\{0,1\}^n$, respectively), and $P_j(x)$ denotes the j-bit long prefix of $G'(x)$. The extreme hybrids correspond to $G'(U_n)$ and $U_{\ell(n)}$, whereas distinguishability of neighboring hybrids can be worked into distinguishability of $G(U_n)$ and U_{n+1}. Loosely speaking, suppose one could distinguish H_n^i from H_n^{i+1}. Then, using $P_j(s) = G_2(s)P_{j-1}(G_1(s))$ (for $j \geq 1$), this means that one can distinguish $H_n^i \equiv (U_i^{(1)}, G_2(U_n^{(2)}), P_{(\ell(n)-i)-1}(G_1(U_n^{(2)})))$ from $H_n^{i+1} \equiv (U_i^{(1)}, U_1^{(1')}, P_{\ell(n)-(i+1)}(U_n^{(2')}))$. Incorporating the generation of $U_i^{(1)}$ and the evaluation of $P_{\ell(n)-i-1}$ into the distinguisher, one could distinguish $(G_1(U_n^{(2)}), G_2(U_n^{(2)})) \equiv G(U_n)$ from $(U_n^{(2')}, U_1^{(1')}) \equiv U_{n+1}$, in contradiction to the pseudorandomness of G. ■

3 How to Construct Pseudorandom Generators

The known constructions transform computation difficulty, in the form of one-way functions (defined below), into pseudorandomness generators. Loosely speaking, a *polynomial-time computable* function is called one-way if any efficient algorithm can invert it only with negligible success probability. For simplicity, we consider only length-preserving one-way functions.

Definition 6 (one-way function): *A* one-way function*, f, is a polynomial-time computable function such that for every probabilistic polynomial-time algorithm A', every*

positive polynomial $p(\cdot)$, and all sufficiently large n's

$$\Pr_{x\sim U_n}\left[A'(f(x))\in f^{-1}(f(x))\right] \;<\; \frac{1}{p(n)}$$

where U_n is the uniform distribution over $\{0,1\}^n$.

Popular candidates for one-way functions are based on the conjectured intractability of integer factorization (cf. [30] for state of the art), the discrete logarithm problem (cf. [31] analogously), and decoding of random linear code [16]. The infeasibility of inverting f yields a weak notion of unpredictability: Let $b_i(x)$ denotes the ith bit of x. Then, for every probabilistic polynomial-time algorithm A (and sufficiently large n), it must be the case that $\Pr_{i,x}[A(i,f(x)) \neq b_i(x)] > 1/2n$, where the probability is taken uniformly over $i \in \{1,...,n\}$ and $x \in \{0,1\}^n$. A stronger (and in fact strongest possible) notion of unpredictability is that of a hard-core predicate. Loosely speaking, a *polynomial-time computable* predicate b is called a hard-core of a function f if any efficient algorithm, given $f(x)$, can guess $b(x)$ only with success probability that is negligible better than half.

Definition 7 (hard-core predicate [4]): *A polynomial-time computable predicate $b : \{0,1\}^* \to \{0,1\}$ is called a* hard-core *of a function f if for every probabilistic polynomial-time algorithm A', every positive polynomial $p(\cdot)$, and all sufficiently large n's*

$$\Pr_{x\sim U_n}[A'(f(x))=b(x)] \;<\; \frac{1}{2}+\frac{1}{p(n)}$$

Clearly, if b is a hard-core of a 1-1 polynomial-time computable function f then f must be one-way.[4] It turns out that any one-way function can be slightly modified so that it has a hard-core predicate.

Theorem 8 (A generic hard-core [13]): *Let f be an arbitrary one-way function, and let g be defined by $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$, where $|x|=|r|$. Let $b(x,r)$ denote the inner-product mod 2 of the binary vectors x and r. Then the predicate b is a hard-core of the function g.*

See proof in [11, Apdx C.2]. We are now ready to present constructions of pseudorandom generators.

3.1 The Preferred Presentation

In view of Theorem 5, we may focus on constructing pseudorandom generators with stretch function $\ell(n) = n+1$. Such a construction is presented next.

Proposition 9 (A simple construction of pseudorandom generators): *Let b be a hard-core predicate of a polynomial-time computable 1-1 function f. Then, $G(s) \stackrel{\text{def}}{=} f(s)\,b(s)$ is a pseudorandom generator.*

[4] Functions that are not 1-1 may have hard-core predicates of information-theoretic nature; but these are of no use to us here. For example, functions of the form $f(\sigma,x) = 0f'(x)$ (for $\sigma \in \{0,1\}$) have an "information theoretic" hard-core predicate $b(\sigma,x) = \sigma$.

Proof Sketch: Clearly the $|s|$-bit long prefix of $G(s)$ is uniformly distributed (since f is 1-1 and onto $\{0,1\}^{|s|}$). Hence, the proof boils down to showing that distinguishing $f(s)b(s)$ from $f(s)\sigma$, where σ is a random bit, yields contradiction to the hypothesis that b is a hard-core of f (i.e., that $b(s)$ is *unpredictable* from $f(s)$). Intuitively, such a distinguisher also distinguishes $f(s)b(s)$ from $f(s)\overline{b(s)}$, where $\overline{\sigma} = 1-\sigma$, and so yields an algorithm for predicting $b(s)$ based on $f(s)$. ■

In a sense, the key point in the above proof is showing that the unpredictability of the output of G implies its pseudorandomness. The fact that (next bit) unpredictability and pseudorandomness are equivalent in general is proven explicitly in the alternative presentation below.

3.2 An Alternative Presentation

The above presentation is different but analogous to the original construction of pseudorandom generators suggested by Blum and Micali [4]: Given an arbitrary stretch function $\ell:\mathbb{N}\to\mathbb{N}$, a 1-1 one-way function f with a hard-core b, one defines

$$G(s) \stackrel{\text{def}}{=} b(x_0)b(x_1)\cdots b(x_{\ell(|s|)-1})\,,$$

where $x_0 = s$ and $x_i = f(x_{i-1})$ for $i = 1, ..., \ell(|s|)-1$. The pseudorandomness of G is established in two steps, using the notion of (next bit) unpredictability. An ensemble $\{Z_n\}_{n\in\mathbb{N}}$ is called unpredictable if any probabilistic polynomial-time machine obtaining a prefix of Z_n fails to predict the next bit of Z_n with probability non-negligibly higher than $1/2$.

Step 1: One first proves that the ensemble $\{G(U_n)\}_{n\in\mathbb{N}}$, where U_n is uniform over $\{0,1\}^n$, is (next-bit) unpredictable (from right to left) [4].
Loosely speaking, if one can predict $b(x_i)$ from $b(x_{i+1})\cdots b(x_{\ell(|s|)-1})$ then one can predict $b(x_i)$ given $f(x_i)$ (i.e., by computing $x_{i+1}, ..., x_{\ell(|s|)-1}$, and so obtaining $b(x_{i+1})\cdots b(x_{\ell(|s|)})$). But this contradicts the hard-core hypothesis.

Step 2: Next, one uses Yao's observation by which a (polynomial-time constructible) ensemble is *pseudorandom if and only if it is* (next-bit) *unpredictable* (cf. [10, Sec. 3.3.4]).
Clearly, if one can predict the next bit in an ensemble then one can distinguish this ensemble from the uniform ensemble (which in unpredictable regardless of computing power). However, here we need the other direction which is less obvious. Still, one can show that (next bit) unpredictability implies indistinguishability from the uniform ensemble. Specifically, consider the following "hybrid" distributions, where the ith hybrid takes the first i bits from the questionable ensemble and the rest from the uniform one. Thus, distinguishing the extreme hybrids implies distinguishing some neighboring hybrids, which in turn implies next-bit predictability (of the questionable ensemble).

3.3 A General Condition for the Existence of Pseudorandom Ggenerators

Recall that given any one-way 1-1 function, we can easily construct a pseudorandom generator. Actually, the 1-1 requirement may be dropped, but the currently known construction – for the general case – is quite complex. Still we do have.

Theorem 10 (On the existence of pseudorandom generators [18]):
Pseudorandom generators exist if and only if one-way functions exist.

To show that the existence of pseudorandom generators imply the existence of one-way functions, consider a pseudorandom generator G with stretch function $\ell(n) = 2n$. For $x, y \in \{0,1\}^n$, define $f(x,y) \stackrel{\text{def}}{=} G(x)$, and so f is polynomial-time computable (and length-preserving). It must be that f is one-way, or else one can distinguish $G(U_n)$ from U_{2n} by trying to invert and checking the result: Inverting f on its range distribution refers to the distribution $G(U_n)$, whereas the probability that U_{2n} has inverse under f is negligible.

The interesting direction is the construction of pseudorandom generators based on any one-way function. In general (when f may not be 1-1) the ensemble $f(U_n)$ may not be pseudorandom, and so Construction 9 (i.e., $G(s) = f(s)b(s)$, where b is a hard-core of f) cannot be used *directly*. One idea of [18] is to hash $f(U_n)$ to an almost uniform string of length related to its entropy, using Universal Hash Functions [5]. (This is done after guaranteeing, that the logarithm of the probability mass of a value of $f(U_n)$ is typically close to the entropy of $f(U_n)$.)[5] But "hashing $f(U_n)$ down to length comparable to the entropy" means shrinking the length of the output to, say, $n' < n$. This foils the entire point of stretching the n-bit seed. Thus, a second idea of [18] is to compensate for the $n - n'$ loss by extracting these many bits from the seed U_n itself. This is done by hashing U_n, and the point is that the $(n - n' + 1)$-bit long hash value does not make the inverting task any easier. Implementing these ideas turns out to be more difficult than it seems, and indeed an alternative construction would be most appreciated.

4 Pseudorandom Functions

Pseudorandom generators allow to efficiently generate long pseudorandom sequences from short random seeds. Pseudorandom functions (defined below) are even more powerful: They allow efficient direct access to a huge pseudorandom sequence (which is infeasible to scan bit-by-bit). Put in other words, pseudorandom functions can replace truly random functions in any efficient application (e.g., most notably in cryptography). That is, pseudorandom functions are indistinguishable from random functions by efficient machines that may obtain the function values at arguments of their choice. (Such machines are called oracle machines, and if M is such machine and f is a function, then $M^f(x)$ denotes the computation of M on input x when M's queries are answered by the function f.)

Definition 11 (pseudorandom functions [12]): *A* pseudorandom function (ensemble)*, with length parameters $\ell_{\mathrm{D}}, \ell_{\mathrm{R}} : \mathbb{N} \to \mathbb{N}$, is a collection of functions $F \stackrel{\text{def}}{=} \{f_s : \{0,1\}^{\ell_{\mathrm{D}}(|s|)} \to \{0,1\}^{\ell_{\mathrm{R}}(|s|)}\}_{s \in \{0,1\}^*}$ satisfying*

[5] Specifically, given an arbitrary one way function f', one first constructs f by taking a "direct product" of sufficiently many copies of f'. For example, for $x_1, ..., x_{n^2} \in \{0,1\}^n$, we let $f(x_1, ..., x_{n^2}) \stackrel{\text{def}}{=} f'(x_1), ..., f'(x_{n^2})$.

- (efficient evaluation): *There exists an efficient* (deterministic) *algorithm that given a* seed, s, *and an* $\ell_{\mathrm{D}}(|s|)$*-bit* argument, x, *returns the* $\ell_{\mathrm{R}}(|s|)$*-bit long value* $f_s(x)$. (Thus, the seed s is an "effective description" of the function f_s.)
- (pseudorandomness): *For every probabilistic polynomial-time oracle machine,* M, *for every positive polynomial* p *and all sufficiently large* n*'s*

$$\left| \Pr_{f\sim F_n}[M^f(1^n)=1] - \Pr_{\rho\sim R_n}[M^\rho(1^n)=1] \right| < \frac{1}{p(n)}$$

where F_n *denotes the distribution on* $f_s \in F$ *obtained by selecting* s *uniformly in* $\{0,1\}^n$, *and* R_n *denotes the uniform distribution over all functions mapping* $\{0,1\}^{\ell_{\mathrm{D}}(n)}$ *to* $\{0,1\}^{\ell_{\mathrm{R}}(n)}$.

Suppose, for simplicity, that $\ell_{\mathrm{D}}(n) = n$ and $\ell_{\mathrm{R}}(n) = 1$. Then a function uniformly selected among 2^n functions (of a pseudorandom ensemble) presents an input-output behavior that is indistinguishable in $\mathrm{poly}(n)$-time from the one of a function selected at random among all the 2^{2^n} Boolean functions. Contrast this with the 2^n pseudorandom sequences, produced by a pseudorandom generator, that are computationally indistinguishable from a sequence selected uniformly among all the $2^{\mathrm{poly}(n)}$ many sequences. Still pseudorandom functions can be constructed from any pseudorandom generator.

Theorem 12 (How to construct pseudorandom functions [12]): *Let* G *be a pseudorandom generator with stretching function* $\ell(n) = 2n$. *Let* $G_0(s)$ (*resp.,* $G_1(s)$) *denote the first* (*resp., last*) $|s|$ *bits in* $G(s)$, *and*

$$G_{\sigma_{|s|}\cdots\sigma_2\sigma_1}(s) \stackrel{\mathrm{def}}{=} G_{\sigma_{|s|}}(\cdots G_{\sigma_2}(G_{\sigma_1}(s))\cdots)$$

Then, the function ensemble $\{f_s : \{0,1\}^{|s|} \to \{0,1\}^{|s|}\}_{s\in\{0,1\}^*}$, *where* $f_s(x) \stackrel{\mathrm{def}}{=} G_x(s)$, *is pseudorandom with length parameters* $\ell_{\mathrm{D}}(n) = \ell_{\mathrm{R}}(n) = n$.

The above construction can be easily adapted to any (polynomially-bounded) length parameters $\ell_{\mathrm{D}}, \ell_{\mathrm{R}} : \mathbb{N} \to \mathbb{N}$. We mention that pseudorandom functions have been used to derive negative results in computational learning theory [35] and in complexity theory (e.g., in the context of Natural Proofs [32]).

5 Further Discussion of Pseudorandom Generators

In this section we discuss some of the applications and conceptual aspects of pseudorandom generators.

5.1 The Applicability of Pseudorandom Generators

Randomness is playing an increasingly important role in computation: It is frequently used in the design of sequential, parallel and distributed algorithms, and is of course central to cryptography. Whereas it is convenient to design such algorithms making free

use of randomness, it is also desirable to minimize the usage of randomness in real implementations (since generating perfectly random bits via special hardware is quite expensive). Thus, pseudorandom generators (as defined above) are a key ingredient in an "algorithmic tool-box" – they provide an automatic compiler of programs written with free usage of randomness into programs that make an economical use of randomness.

Indeed, "pseudo-random number generators" have appeared with the first computers. However, typical implementations use generators that are not pseudorandom according to the above definition. Instead, at best, these generators are shown to pass SOME ad-hoc statistical test (cf. [20]). We warn that the fact that a "pseudo-random number generator" passes some statistical tests, does not mean that it will pass a new test and that it is good for a future (untested) application. Furthermore, the approach of subjecting the generator to some ad-hoc tests fails to provide general results of the type stated above (i.e., of the form "for ALL practical purposes using the output of the generator is as good as using truly unbiased coin tosses"). In contrast, the approach encompassed in Definition 2 aims at such generality, and in fact is tailored to obtain it: The notion of computational indistinguishability, which underlines Definition 2, covers all possible efficient applications postulating that for all of them pseudorandom sequences are as good as truly random ones.

Pseudorandom generators and functions are of key importance in Cryptography. They are typically used to establish private-key encryption and authentication schemes (cf. [11, Sec. 1.5.2 & 1.6.2]). For example, suppose that two parties share a random n-bit string, s, specifying a pseudorandom function (as in Definition 11), and that s is unknown to the adversary. Then, these parties may send encrypted messages to one another by XORing the message with the value of f_s at a random point. That is, to encrypt $m \in \{0,1\}^{\ell_{\mathrm{R}}(n)}$, the sender uniformly selects $r \in \{0,1\}^{\ell_{\mathrm{D}}(n)}$, and sends $(r, m \oplus f_s(r))$ to the receiver. Note that the security of this encryption scheme relies on the fact that, for *every* computationally-feasible adversary (not only to adversary strategies that were envisioned and tested), the values of the function f_s on such r's look random.

5.2 The Intellectual Contents of Pseudorandom Generators

We shortly discuss some intellectual aspects of pseudorandom generators as defined above.

Behavioristic versus Ontological. Our definition of pseudorandom generators is based on the notion of computational indistinguishability. The behavioristic nature of the latter notion is best demonstrated by confronting it with the Kolmogorov-Chaitin approach to randomness. Loosely speaking, a string is *Kolmogorov-random* if its length equals the length of the shortest program producing it. This shortest program may be considered the "true explanation" to the phenomenon described by the string. A Kolmogorov-random string is thus a string that does not have a substantially simpler (i.e., shorter) explanation than itself. Considering the simplest explanation of a phenomenon may be viewed as an ontological approach. In contrast, considering the effect of phenomena (on an observer), as underlying the definition of pseudorandomness, is a behavioristic approach. Furthermore, there exist probability distributions that are not uniform (and are not even statistically close to a uniform distribution) but nevertheless are indistinguishable from

a uniform distribution by any efficient procedure [37, 15]. Thus, distributions that are ontologically very different, are considered equivalent by the behavioristic point of view taken in the definitions above.

A relativistic view of randomness. Pseudorandomness is defined above in terms of its observer. It is a distribution that cannot be told apart from a uniform distribution by any efficient (i.e. polynomial-time) observer. However, pseudorandom sequences may be distinguished from random ones by infinitely powerful computers (not at our disposal!). Specifically, an exponential-time machine can easily distinguish the output of a pseudorandom generator from a uniformly selected string of the same length (e.g., just by trying all possible seeds). Thus, pseudorandomness is subjective to the abilities of the observer.

Randomness and Computational Difficulty. Pseudorandomness and computational difficulty play dual roles: The definition of pseudorandomness relies on the fact that putting computational restrictions on the observer gives rise to distributions that are not uniform and still cannot be distinguished from uniform. Furthermore, the construction of pseudorandom generators rely on conjectures regarding computational difficulty (i.e., the existence of one-way functions), and this is inevitable: given a pseudorandom generator, we can construct one-way functions. Thus, (non-trivial) pseudorandomness and computational hardness can be converted back and forth.

6 A General Paradigm

Pseudorandomness as surveyed above can be viewed as an important special case of a general paradigm. A generic formulation of pseudorandom generators consists of specifying three fundamental aspects – the *stretching measure* of the generators; the class of distinguishers that the generators are supposed to fool (i.e., the algorithms with respect to which the *computational indistinguishability* requirement should hold); and the resources that the generators are allowed to use (i.e., their own *computational complexity*). In the above presentation we focused on polynomial-time generators (thus having polynomial stretching measure) that fool any probabilistic polynomial-time observers. A variety of other cases are of interest too, and we briefly discuss some of them.

6.1 Weaker Notions of Computational Indistinguishability

Whenever the aim is to replace random sequences utilized by an algorithm with pseudorandom ones, one may try to capitalize on knowledge of the target algorithm. Above we have merely used the fact that the target algorithm runs in polynomial-time. However, if the application utilizes randomness in a restricted way then feeding it with sequences of lower "randomness-quality" may do. For example, if we know that the algorithm uses very little work-space then we may use weaker forms of pseudorandom generators, which may be easier to construct, that suffice to fool bounded-space distinguishers. Similarly, very weak forms of pseudorandomness suffice for randomized algorithms that can be analyzed when only referring to some specific properties of the random sequence

they uses (e.g., pairwise independence of elements of the sequence). In general, weaker notions of computational indistinguishability such as fooling space-bounded algorithms, constant-depth circuits, and even specific tests (e.g., testing pairwise independence of the sequence), arise naturally, and generators producing sequences that fool such distinguishers are useful in a variety of applications. Needless to say that we advocate a rigorous formulation of the characteristics of such applications and rigorous constructions of generators that fool the type of distinguishers that emerge. We mention some results of this type.

Fooling space-bounded algorithms. Here we consider space-bounded randomized algorithms that have on-line access to their random-tape, and so the potential distinguishers have on-line access to the input that they inspect. Two main results in this area are:

Theorem 13 ($\mathcal{RL} \subseteq \mathcal{SC}$ [26, 27]): *Any language decidable by a log-space randomized algorithm is decidable by a polynomial-time deterministic algorithm of poly-logarithmic space complexity.*

Theorem 14 (The Nisan–Zuckerman Generator [29]): *Any language decidable by a linear-space polynomial-time randomized algorithm is decidable by a randomized algorithm of the same complexities that uses only a linear number of coin tosses.*

Both theorems are actually special cases of more general results that refer to arbitrary computations (rather than to decision problems).

Fooling constant-depth circuits. As a special case, we consider the problem of approximately counting the number of satisfying assignments of a DNF formula. Put in other words, we wish to generate "pseudorandom" sequences that are as likely to satisfy a given DNF formula as uniformly selected sequences. Nisan showed that such "pseudorandom" sequences can be produced using seeds of polylogarithmic length [25]. By trying all possible seeds, one can approximately count the number of satisfying assignments of a DNF formula in deterministic quasi-polynomial time.

Pairwise independent generators. We consider distributions of n-long sequences over a finite set S. For $t \in \mathbb{N}$, such a distribution is called t-wise independent if its projection on any t coordinates yields a distribution that is uniform over S^t. We focus on the case where $|S|$ is a prime power, and so S can be identified with a finite field F. In such a case, given $1^n, 1^t$ and a representation of the field F so that $|F| > n$, one can generated a t-wise independent distribution over F^n in polynomial-time, using a random seed of length $t \cdot \log_2 |F|$. Specifically, the seed is used to specify a polynomial of degree $t-1$ over F, and the ith element in the output sequence is the result of evaluating this polynomial at the ith field element (cf. [2, 7]).

Small-bias generators. Here, we consider distributions of n-long sequences over $\{0,1\}$. For $\epsilon \in [0,1]$, such a distribution is called ϵ-bias if for every non-empty subset I, the exclusive-or of the bits at locations I equals 1 with probability at least $(1-\epsilon) \cdot \frac{1}{2}$ and at most $(1+\epsilon) \cdot \frac{1}{2}$.

Theorem 15 (small-bias generators [24]): *Given n and ϵ, one can generate an ϵ-bias distribution over $\{0,1\}^n$ in $\mathrm{poly}(n, \log(1/\epsilon))$-time, using a random seed of length $O(\log(n/\epsilon))$.*

See [11, Sec. 3.6.2] for more details.

Samplers (and hitters) and extractors (and dispersers). Here we consider an arbitrary function $\nu : \{0,1\}^n \to [0,1]$, and seeks a universal procedure for approximating the average value of ν, denoted $\overline{\nu}$ (i.e., $\overline{\nu} \stackrel{\text{def}}{=} 2^{-n} \sum_x \nu(x)$). Such a (randomized) procedure is called a sampler. It is given three parameters, n, ϵ and δ, as well as oracle access to ν, and needs to output a value $\widetilde{\nu}$ so that $\Pr[|\overline{\nu} - \widetilde{\nu}| > \epsilon] < \delta$. A hitter is given the parameters, n, ϵ and δ, as well as a value v so that $|\{x : \nu(x) = v\}| > \epsilon \cdot 2^n$ (and oracle access to ν), and is required to find, with probability at least $1 - \delta$, a preimage x so that $\nu(x) = v$. A sampler is called non-adaptive if it determines its queries based only on its internal coin tosses (i.e., independently on the answers obtained for previous queries); it is called oblivious if its output is a predetermined function of the sequence of oracle answers; and it is called averaging if its output equals the average value of the oracle answers. (In a sense, a non-adaptive sampler corresponds to a "pseudorandom generator" that produces at random a sequence of queries that, with high probability, needs to be "representative" of the average value of any function.) We focus on the randomness and query complexities of samplers, and mention that any sampler yields a hitter with identical complexities.

Theorem 16 (The Median-of-Averages Sampler [3]): *There exists a polynomial-time* (oblivious) *sampler of randomness complexity $O(n + \log(1/\delta))$ and query complexity $O(\epsilon^{-2} \log(1/\delta))$. Specifically, the sampler outputs the median value among $O(\log(1/\delta))$ values, where each of these values is the average of $O(\epsilon^{-2})$ distinct oracle answers.*[6]

The randomness complexity can be further reduced to $n + O(\log(1/\epsilon\delta))$, and both complexities are optimal up-to a constant multiplicative factor; see [11, Sec. 3.6.4]. Averaging samplers are closely related to extractors, but the study of the latter tends to focus more closely on the randomness complexity (and allow query complexity that is polynomial in the above).[7] A function $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^m$ is called a (k, ϵ)-extractor if for any random variable X so that $\max_x\{\Pr[X = x]\} \leq 2^{-k}$ it holds that the statistical difference between $E(X, U_t)$ and U_m is at most ϵ, where U_t and U_m are independently and uniformly distributed over $\{0,1\}^t$ and $\{0,1\}^m$, respectively. (An averaging sampler of randomness complexity $r(m, \epsilon, \delta)$ and query complexity $q(m, \epsilon, \delta)$ corresponds to an extractor in which (the yet unspecified parameters are) $n = r(m, \epsilon, \delta)$, $t = \log_2 q(m, \epsilon, \delta)$, and $k = n - \log_2(1/\delta)$.) A landmark in the study of extractors is the following

Theorem 17 (Trevisan's Extractor [36]): *For any $a, b > 0$, let $k(n) = n^a$ and $m(n) = \lceil k(n)^{1-b} \rceil$. For $t(n) = O(\log n)$ and $\epsilon(n) > 1/\mathrm{poly}(n)$, there exists a polynomial-time*

[6] Each of the $O(\log(1/\delta))$ sequences of $O(\epsilon^{-2})$ queries is produced by a pairwise independent generator, and the seeds used for these different sequences are generated by a random walk on an expander graph (cf. [1] and [11, Sec. 3.6.3]).

[7] The relation between hitters and dispersers is analogous.

computable family of functions $\{E_n : \{0,1\}^n \times \{0,1\}^{t(n)} \to \{0,1\}^{m(n)}\}_{n\in\mathbb{N}}$ *so that* E_n *is an* $(k(n), \epsilon(n))$*-extractor.*

The theorem is proved by reducing the construction of extractors to the construction of certain pseudorandom generators (considered in the next subsection). The reduction is further discussed at the end of the next subsection.

6.2 Alternative Notions of Generator Efficiency

The above discussion has focused on one aspect of the pseudorandomness question – the resources or type of the observer (or potential distinguisher). Another important question is *at what cost* can pseudorandom sequences be generated (from much shorter seeds, assuming this is at all possible). Throughout this survey we have required the generation process to be at least as efficient as the efficiency limitations of the distinguisher.[8] This seems indeed "fair" and natural. Allowing the generator to be more complex (i.e., use more time or space resources) than the distinguisher seems unfair (and is typically unreasonable in the context of cryptography), but still yields interesting consequences in the context of "de-randomization" (i.e., transforming randomized algorithms into equivalent deterministic algorithms (of slightly higher complexity)). For example, one may consider generators working in time exponential in the length of the seed. As observed by Nisan and Wigderson [28], in some cases we lose nothing by being more liberal (i.e., allowing exponential-time generators). To see why, we consider a typical de-randomization argument, proceeding in two steps: First one replaces the true randomness of the algorithm by pseudorandom sequences generated from much shorter seeds, and next one goes deterministically over all possible seeds and looks for the most frequent behavior of the modified algorithm. Thus, in such a case the deterministic complexity is anyhow exponential in the seed length. The benefit of allowing exponential-time generators is that constructing exponential-time generators may be easier than constructing polynomial-time ones. A typical result in this vein follows.

Theorem 18 (De-randomization of BPP [19] (building upon [28])): *Suppose that there exists a language* $L \in \mathcal{E}$ *having almost-everywhere exponential circuit complexity.*[9] *Then,* $\mathcal{BPP} = \mathcal{P}$.

Proof Sketch: Underlying the proof is a construction of a pseudorandom generator due to Nisan and Wigderson [25, 28]. This construction utilizes a predicate computable in exponential-time but unpredictable, even to within a particular exponential advantage, by any circuit family of a particular exponential size. (The crux of [19] is in supplying

[8] If fact, we have required the generator to be more efficient than the distinguisher: The former was required to be a fixed polynomial-time algorithm, whereas the latter was allowed to be any algorithm with polynomial running time.

[9] We say that L is in $\mathcal{E}$ if there exists an exponential time algorithm for deciding L; that is, the running-time of the algorithm on input x is at most $2^{O(|x|)}$. By saying that L has almost-everywhere exponential circuit complexity we mean that there exists a constant $b > 0$ such that, for all but finitely many k's, any circuit C_k that correctly decides L on $\{0,1\}^k$ has size at least 2^{bk}.

such a predicate, given the hypothesis.) Given such a predicate the generator works by evaluating the predicate on exponentially-many subsequences of the bits of the seed so that the intersection of any two subsets is relatively small.[10] Thus, for some constant $b > 0$ and all k's, the generator stretches seeds of length k into sequences of length 2^{bk} that (as loosely argued below) cannot be distinguished from truly random sequences by any circuit of size 2^{bk}. The de-randomization of $\mathcal{BPP}$ proceeds by setting the seed-length to be logarithmic in the input length, and utilizing the above generator.

The above generator fools circuits of the stated size, even when these circuits are presented with the seed as auxiliary input. (These circuits are smaller than the running time of the generator and so they cannot just evaluate the generator on the given seed.) The proof that the generator fools such circuits refers to the characterization of pseudorandom sequences as unpredictable ones. Thus, one proves that the next bit in the generator's output cannot be predicted given all previous bits (as well as the seed). Assuming that a small circuit can predict the next bit of the generator, we construct a circuit for predicting the hard predicate. The new circuit incorporates the best (for such prediction) augmentation of the input to the circuit into a seed for the generator (i.e., the bits not in the specific subset of the seed are fixed in the best way). The key observation is that all other bits in the output of the generator depend only on a small fraction of the input bits (i.e., recall the small intersection clause above), and so circuits for computing these other bits have relatively small size (and so can be incorporated in the new circuit). Using all these circuits, the new circuit forms the adequate input for the next-bit predicting circuit, and outputs whatever the latter circuit does. ■

Connection to extractors. Trevisan's construction [36] adapts the computational framework underlying the Nisan–Wigderson Generator [28] to the information-theoretic context of extractors. His adaptation is based on two key observations. The first observation is that the generator itself uses a (supposedly hard) predicate as a black-box. Trevisan's construction utilizes a "random" predicate which is encoded by the first input to the extractor. For example, the n-bit input may encode a predicate on $\log_2 n$ bits in the obvious manner. The second input to the extractor, having length $t = O(\log n)$, will be used as the seed to the resulting generator (defined by using this random predicate in a black-box manner). The second key observation is that the proof of indistinguishability of the generator provides a black-box procedure for computing the underlying predicate when given oracle access to a distinguisher. Thus, any subset $S \subset \{0,1\}^m$ of the possible outputs of the extractor gives rise to a relatively small set P_S of predicates, so that for each value $x \in \{0,1\}^n$ of the first input to the extractor, if S "distinguishes" the output of the extractor (on a random second input) from the uniform distribution then one of the predicates in P_S equals the predicate associated with x. It follows that for every set S, the set of possible first inputs for which the probability that the extractor hits S does not approximate the density of S is small. This establishes the extraction property.

[10] These subsets have size linear in the length of the seed, and intersect on a constant fraction of their respective size. Furthermore, they can be determined within exponential-time.

References

1. M. Ajtai, J. Komlos, E. Szemerédi. Deterministic Simulation in LogSpace. In *19th ACM Symposium on the Theory of Computing*, pages 132–140, 1987.
2. N. Alon, L. Babai and A. Itai. A fast and Simple Randomized Algorithm for the Maximal Independent Set Problem. *J. of Algorithms*, Vol. 7, pages 567–583, 1986.
3. M. Bellare, O. Goldreich, and S. Goldwasser. Randomness in Interactive Proofs. *Computational Complexity*, Vol. 4, No. 4, pages 319–354, 1993.
4. M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SIAM Journal on Computing*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd IEEE Symposium on Foundations of Computer Science*, 1982.
5. L. Carter and M. Wegman. Universal Hash Functions. *Journal of Computer and System Science*, Vol. 18, 1979, pages 143–154.
6. G.J. Chaitin. On the Length of Programs for Computing Finite Binary Sequences. *Journal of the ACM*, Vol. 13, pages 547–570, 1966.
7. B. Chor and O. Goldreich. On the Power of Two–Point Based Sampling. *Jour. of Complexity*, Vol 5, 1989, pages 96–106. Preliminary version dates 1985.
8. B. Chor and O. Goldreich. Unbiased Bits from Sources of Weak Randomness and Probabilistic Communication Complexity. *SIAM Journal on Computing*, Vol. 17, No. 2, pages 230–261, 1988.
9. T.M. Cover and G.A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Inc., New-York, 1991.
10. O. Goldreich. *Foundation of Cryptography – Fragments of a Book*. February 1995. Available from `http://theory.lcs.mit.edu/~oded/frag.html`.
11. O. Goldreich. *Modern Cryptography, Probabilistic Proofs and Pseudorandomness.* Algorithms and Combinatorics series (Vol. 17), *Springer*, 1998.
12. O. Goldreich, S. Goldwasser, and S. Micali. How to Construct Random Functions. *Journal of the ACM*, Vol. 33, No. 4, pages 792–807, 1986.
13. O. Goldreich and L.A. Levin. Hard-core Predicates for any One-Way Function. In *21st ACM Symposium on the Theory of Computing*, pages 25–32, 1989.
14. O. Goldreich and S. Micali. Increasing the Expansion of Pseudorandom Generators. Unpublished manuscript, 1984.
15. O. Goldreich, and H. Krawczyk. On Sparse Pseudorandom Ensembles. *Random Structures and Algorithms*, Vol. 3, No. 2, (1992), pages 163–174.
16. O. Goldreich, H. Krawcyzk and M. Luby. On the Existence of Pseudorandom Generators. *SIAM Journal on Computing*, Vol. 22-6, pages 1163–1175, 1993.
17. S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Science*, Vol. 28, No. 2, pages 270–299, 1984. Preliminary version in *14th ACM Symposium on the Theory of Computing*, 1982.
18. J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, Volume 28, Number 4, pages 1364–1396, 1999. Preliminary versions by Impagliazzo et. al. in *21st ACM Symposium on the Theory of Computing* (1989) and Håstad in *22nd ACM Symposium on the Theory of Computing* (1990).
19. R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *29th ACM Symposium on the Theory of Computing*, pages 220–229, 1997.
20. D.E. Knuth. *The Art of Computer Programming*, Vol. 2 (*Seminumerical Algorithms*). Addison-Wesley Publishing Company, Inc., 1969 (first edition) and 1981 (second edition).

21. A. Kolmogorov. Three Approaches to the Concept of "The Amount Of Information". *Probl. of Inform. Transm.*, Vol. 1/1, 1965.
22. L.A. Levin. Randomness Conservation Inequalities: Information and Independence in Mathematical Theories. *Inform. and Control*, Vol. 61, pages 15–37, 1984.
23. M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag, August 1993.
24. J. Naor and M. Naor. Small-bias Probability Spaces: Efficient Constructions and Applications. *SIAM J. on Computing*, Vol 22, 1993, pages 838–856.
25. N. Nisan. Pseudorandom bits for constant depth circuits. *Combinatorica*, Vol. 11 (1), pages 63–70, 1991.
26. N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992.
27. N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *Journal of Computational Complexity*, Vol. 4, pages 1-11, 1994.
28. N. Nisan and A. Wigderson. Hardness vs Randomness. *Journal of Computer and System Science*, Vol. 49, No. 2, pages 149–167, 1994.
29. N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Science*, Vol. 52 (1), pages 43–52, 1996.
30. A.M. Odlyzko. The future of integer factorization. *CryptoBytes* (The technical newsletter of RSA Laboratories), Vol. 1 (No. 2), pages 5-12, 1995. Available from `http://www.research.att.com/~amo`
31. A.M. Odlyzko. Discrete logarithms and smooth polynomials. In *Finite Fields: Theory, Applications and Algorithms*, G. L. Mullen and P. Shiue, eds., Amer. Math. Soc., Contemporary Math. Vol. 168, pages 269–278, 1994. Available from `http://www.research.att.com/~amo`
32. A.R. Razborov and S. Rudich. Natural proofs. *Journal of Computer and System Science*, Vol. 55 (1), pages 24–35, 1997.
33. C.E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. Jour.*, Vol. 27, pages 623–656, 1948.
34. R.J. Solomonoff. A Formal Theory of Inductive Inference. *Inform. and Control*, Vol. 7/1, pages 1–22, 1964.
35. L. Valiant. A theory of the learnable. *Communications of the ACM*, Vol. 27/11, pages 1134–1142, 1984.
36. L. Trevisan. Constructions of Near-Optimal Extractors Using Pseudo-Random Generators. In *31st ACM Symposium on the Theory of Computing*, pages 141–148, 1998.
37. A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

A Bound on the Capacity of Backoff and Acknowledgement-Based Protocols ⋆

Leslie Ann Goldberg[1], Mark Jerrum[2], Sampath Kannan[3], and Mike Paterson[1]

[1] Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK. {leslie,msp}@dcs.warwick.ac.uk

[2] School of Computer Science, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, UK. mrj@dcs.ed.ac.uk

[3] Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA 19104-6389 USA. kannan@central.cis.upenn.edu

Abstract. We study contention-resolution protocols for multiple-access channels. We show that *every* backoff protocol is transient if the arrival rate, λ, is at least 0.42 and that the capacity of every backoff protocol is at most 0.42. Thus, we show that backoff protocols have (provably) smaller capacity than full-sensing protocols. Finally, we show that the corresponding results, with the larger arrival bound of 0.531, also hold for every acknowledgement-based protocol.

1 Introduction

A *multiple-access channel* is a broadcast channel that allows multiple users to communicate with each other by sending messages onto the channel. If two or more users simultaneously send messages, then the messages interfere with each other (collide), and the messages are not transmitted successfully. The channel is not centrally controlled. Instead, the users use a contention-resolution protocol to resolve collisions. Thus, after a collision, each user involved in the collision waits a random amount of time (which is determined by the protocol) before re-sending.

Following previous work on multiple-access channels, we work in a *time-slotted* model in which time is partitioned into discrete *time steps.* At the beginning of each time step, a random number of messages enter the system, each of which is associated with a new user which has no other messages to send. The number of messages that enter the system is drawn from a Poisson distribution with mean λ. During each time step, each message chooses independently whether to send to the channel. If *exactly* one message sends to the channel during the time step, then this message leaves the system and we call this a *success.* Otherwise, all of the messages remain in the system and the next time

⋆ A full version appears at http://www.dcs.warwick.ac.uk/~leslie/papers/gjkp.ps. This work was partially supported by EPSRC grant GR/L60982, NSF Grant CCR9820885, and ESPRIT Projects ALCOM-FT and RAND-APX.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 705–717, 2000.
© Springer-Verlag Berlin Heidelberg 2000

step is started. Note that when a message sends to the channel this may or may not result in a success, depending on whether or not any other messages send to the channel.

The quality of a protocol can be measured in several ways. Typically, one models the execution of the protocol as a Markov chain. If the protocol is good (for a given arrival rate λ), the corresponding Markov chain will be *recurrent* (with probability 1, it will eventually return to the empty state in which no messages are waiting). Otherwise, the chain is said to be *transient* (and we also say that a protocol is transient). Note that transience is a very strong form of instability. In particular, if we focus on any finite set of "good" states then if the chain is transient, the probability of visiting these states at least N times during the infinite run of the protocol is exponentially small in N. (This follows because the relevant Markov chain is irreducible and aperiodic.)

Another way to measure the quality of a protocol is to measure its *capacity.* A protocol is said to achieve *throughput* λ if, when it is run with input rate λ, the average success rate is λ. The *capacity* of the protocol [4] is the maximum throughput that it achieves.

The protocols that we consider in this paper are *acknowledgement-based* protocols. In the acknowledgement-based model, the only information that a user receives about the state of the system is the history of its own transmissions. An alternative model is the *full-sensing* model, in which *every* user listens to the channel at *every* step, regardless of whether it sends during the step.[1]

One particularly simple and easy-to-implement class of acknowledgement-based protocols is the class of *backoff protocols.* A backoff protocol is a sequence of probabilities $p_0, p_1, \ldots$. If a message has sent unsuccessfully i times before a time-step, then with probability p_i, it sends during the time-step. Otherwise, it does not send. Kelly and MacPhee [12], [13], [16] gave a formula for the *critical arrival rate,* λ^* , of a backoff protocol, which is the minimum arrival rate for which the expected number of successful transmissions that the protocol makes is finite.[2]

Perhaps the best-known backoff protocol is the *binary exponential backoff protocol* in which $p_i = 2^{-i}$. This protocol is the basis of the Ethernet protocol of Metcalfe and Boggs [17].[3] Kelly and MacPhee showed that the critical arrival rate of this protocol is $\ln 2$. Thus, if $\lambda > \ln 2$, then binary exponential backoff achieves only a finite number of successful transmissions (in expectation). Aldous [1] showed that the binary exponential backoff protocol is not a good protocol for

[1] In practice, it is possible to implement the full-sensing model when there is a single channel, but this becomes increasingly difficult in situations where there are multiple shared channels, such as optical networks. Thus, acknowledgement-based protocols are sometimes preferable to full-sensing protocols. For work on contention-resolution in the multiple-channel setting, see [6].

[2] If $\lambda > \lambda^*$, then the expected number of successes is finite, even if the protocol runs forever. They showed that the critical arrival rate is 0 if the expected number of times that a message sends during the first t steps is $\omega(\log t)$.

[3] There are several differences between the "real-life" Ethernet protocol and "pure" binary exponential backoff, but we do not describe these here.

any positive arrival rate λ. In particular, it is transient and the expected number of successful transmissions in t steps is $o(t)$. MacPhee [16] posed the question of whether there exists a backoff protocol which is recurrent for some positive arrival rate λ.

In this paper, we show that there is no backoff protocol which is recurrent for $\lambda \geq 0.42$. (Thus, *every* backoff protocol is transient if $\lambda \geq 0.42$.) Also, every backoff protocol has capacity at most 0.42. As far as we know, our result is the first proof showing that backoff protocols have smaller capacity than full-sensing protocols. In particular, Mosely and Humblet [19] have discovered a full-sensing protocol with capacity 0.48776.[4] Finally, we show that *no* acknowledgement-based protocol is recurrent for $\lambda \geq 0.530045$.

1.1 Related Work

Backoff protocols and acknowledgement-based protocols have also been studied in an n-user model, which combines contention-resolution with queueing. In this model, it is assumed that n users maintain queues of messages, and that new messages arrive at the tails of the queues. At each step, the users use contention-resolution protocols to try to send the messages at the heads of their queues. It turns out that the queues have a stabilising effect, so some protocols (such as "polynomial backoff") which are unstable in our model [13] are stable in the queueing model [11]. We will not describe queueing-model results here, but the reader is referred to [2], [8], [11], [21].

Much work has gone into determining upper bounds on the capacity that can be achieved by a full-sensing protocol. The current best result is due to Tsybakov and Likhanov [23] who have shown that no protocol can achieve capacity higher than 0.568. (For more information, see [4], [9], [14], [15], [18], [22].)

1.2 Improvements

We choose $\lambda = 0.42$ in order to make the proof of Lemma 3 as simple as possible. The lemma seems to be true for λ down to about 0.41 and presumably the parameters A and B could be tweaked to get λ slightly smaller.

2 Markov Chain Background

An irreducible aperiodic Markov chain $X = \{X_0, X_1, \ldots\}$ with a countable state space Ω (see [10]) is *recurrent* if it returns to its start state with probability 1. That is, it is recurrent if for some state i (and therefore, for all i), $\text{Prob}[X_t = i \text{ for some } t \geq 1 \mid X_0 = i] = 1$. Otherwise, X is said to be *transient.* X is *positive recurrent* (or ergodic) if the expected number of steps that it takes

[4] Mosely and Humblet's protocol is a "tree protocol" in the sense of Capetanakis [3] and Tsybakov and Mikhailov [24]. For a simple analysis of the protocol, see [25]. Vvedenskaya and Pinsker have shown how to modify Mosely and Humblet's protocol to achieve an improvement in the capacity (in the seventh decimal place) [26].

before returning to its start state is finite. We use the following theorems which we take from [5].

Theorem 1. *(Fayolle, Malyshev, Menshikov) A time-homogeneous irreducible aperiodic Markov chain X with countable state space Ω is not positive recurrent if there is a function f with domain Ω and there are constants C, d such that*

1. *there is a state x with $f(x) > C$, and a state x with $f(x) \leq C$, and*
2. *$E[f(X_1) - f(X_0) \mid X_0 = x] \geq 0$ for all x with $f(x) > C$, and*
3. *$E[\,|f(X_1) - f(X_0)| \mid X_0 = x] \leq d$ for every state x.*

Theorem 2. *(Fayolle, Malyshev, Menshikov) A time-homogeneous irreducible aperiodic Markov chain X with countable state space Ω is transient if there are a positive function f with domain Ω and positive constants C, d, ε such that*

1. *there is a state x with $f(x) > C$, and a state x with $f(x) \leq C$, and*
2. *$E[f(X_1) - f(X_0) \mid X_0 = x] \geq \varepsilon$ for all x with $f(x) > C$, and*
3. *if $|f(x) - f(y)| > d$ then the probability of moving from x to y in a single move is 0.*

3 Stochastic Domination and Monotonicity

Suppose that X is a Markov chain and that the (countable) state space Ω of the chain is a *partial order* with binary relation $\leq$. If A and B are random variables taking states as values, then B *dominates* A if and only if there is a joint sample space for A and B in which the value of A is always less than or equal to the value of B. Note that there will generally be other joint sample spaces in which the value of A can exceed the value of B. Nevertheless, We write $A \leq B$ to indicate that B dominates A. We say that X is *monotonic* if for any states $x \leq x'$, the next state conditioned on starting at x' dominates the next state conditioned on starting at x. (Formally, $(X_1 \mid X_0 = x')$ dominates $(X_1 \mid X_0 = x)$.)

When an acknowledgement-based protocol is viewed as a Markov chain, the state is just the collection of messages in the system. (Each message is identified by the history of its transmissions.) Thus, the state space is countable and it forms a partial order with respect to the subset inclusion relation $\subseteq$ (for multisets). We say that a protocol is *deletion resilient* [7] if its Markov chain is monotonic with respect to the subset-inclusion partial order.

Observation 3. *Every acknowledgement-based protocol is deletion resilient.*

As we indicated earlier, we will generally assume that the number of messages entering the system at a given step is drawn from a Poisson process with mean λ. However, it will sometimes be useful to consider other message-arrival distributions. If I and I' are message-arrival distributions, we write $I \leq I'$ to indicate that the number of messages generated under I is dominated by the number of messages generated under I'.

Observation 4. *If the acknowledgement-based protocol P is recurrent under message-arrival distribution I' and $I \leq I'$ then P is also recurrent under I.*

4 Backoff Protocols

In this section, we will show that there is no backoff protocol which is recurrent for $\lambda \geq 0.42$. Our method will be to use the "drift theorems" in Section 2. Let $p_0, p_1, \ldots$ be a backoff protocol. Without loss of generality, we can assume $p_0 = 1$, since we can ignore new arrivals until they first send. Let $\lambda = 0.42$. Let X be the Markov chain described in Section 3 which describes the behaviour of the protocol with arrival rate λ. First, we will construct a potential function (Lyapounov function) f which satisfies the conditions of Theorem 1, that is, a potential function which has a bounded positive drift. We will use Theorem 1 to conclude that the chain is not positive recurrent. Next, we will consider the behaviour of the protocol under a *truncated* arrival distribution and we will use Theorem 2 to show that the protocol is transient. Using Observation 4 (domination), we will conclude that the protocol is also transient with Poisson arrivals at rate λ or higher. Finally, we will show that the *capacity* of every backoff protocol is at most 0.42.

We now define some parameters of a state x. Let $k(x)$ denote the number of messages in state x. If $k(x) = 0$, then $p(x) = r(x) = u(x) = 0$. Otherwise, let $m_1, \ldots, m_{k(x)}$ denote the messages in state x, with send probabilities $\rho_1 \geq \cdots \geq \rho_{k(x)}$. Let $p(x) = \rho_1$ and let $r(x)$ denote the probability that at least one of $m_2, \ldots, m_{k(x)}$ sends on the next step. Let $u(x)$ denote the probability that exactly one of $m_2, \ldots, m_{k(x)}$ sends on the next step. Clearly $u(x) \leq r(x)$. If $p(x) < r(x)$ then we use the following (tighter) upper bound for $u(x)$, which we prove in the full version.

Lemma 1. *If $p(x) < r(x)$ then $u(x) \leq \frac{r(x) - r(x)^2/2}{1 - p(x)/2}$.*

Let $\mathcal{S}(x)$ denote the probability that there is a success when the system is run for one step starting in state x. (Recall that a success occurs if *exactly* one message sends during the step. This single sender might be a new arrival, or it might be an old message from state x.) Let

$$g(r,p) = e^{-\lambda}[(1-r)p + (1-p)\min\{r, \frac{r - r^2/2}{1 - p/2}\} + (1-p)(1-r)\lambda].$$

We now have the following corollary of Lemma 1.

Corollary 1. *For any state x, $\mathcal{S}(x) \leq g(r(x), p(x))$.*

Let $s(x)$ denote the probability that at least one message in state x sends on the next step. That is, $s(x)$ is the probability that at least one *existing* message in x sends. New arrivals may also send. There may or may not be a success. (Thus, if x is the empty state, then $s(x) = 0$.) Let $A = 0.9$ and $B = 0.41$. For every $\pi \in [0,1]$, let $c(\pi) = \max(0, -A\,\pi + B)$. For every state x, let $f(x) = k(x) + c(s(x))$. The function f is the potential function alluded to earlier, which plays a leading role in Theorems 1 and 2. To a first approximation, $f(x)$ counts the number of messages in the state x, but the small correction term is crucial. Finally, let

$$h(r,p) = \lambda - g(r,p) - [1 - e^{-\lambda}(1-p)(1-r)(1+\lambda)]c(r + p - r\,p) + e^{-\lambda}p(1-r)c(r).$$

Now we have the following.

Observation 5. *For any state x, $E[\,|f(X_1) - f(X_0)| \mid X_0 = x] \leq 1 + B$.*

Lemma 2. *For any state x, $E[f(X_1) - f(X_0) \mid X_0 = x] \geq h(r(x), p(x))$.*

Proof. The result follows from the following chain of inequalities, each link of which is justified below.

$$\begin{aligned} E[f(X_1) - f(X_0) \mid X_0 = x] &= \lambda - \mathcal{S}(x) + E[c(s(X_1)) \mid X_0 = x] - c(s(x)) \\ &\geq \lambda - g(r(x), p(x)) + E[c(s(X_1)) \mid X_0 = x] - c(s(x)) \\ &\geq \lambda - g(r(x), p(x)) + e^{-\lambda}(1 - p(x))(1 - r(x))(1 + \lambda)c(s(x)) \\ &\qquad + e^{-\lambda}p(x)(1 - r(x))c(r(x)) - c(s(x)) \\ &= h(r(x), p(x)). \end{aligned}$$

The first inequality follows from Corollary 1. The second comes from substituting exact expressions for $c(s(X_1))$ whenever the form of X_1 allows it, and using the bound $c(s(X_1)) \geq 0$ elsewhere. If none of the existing messages sends and there is at most one arrival, then $c(s(X_1)) = c(s(x))$, giving the third term; if message m_1 alone sends and there are no new arrivals then $c(s(X_1)) = c(r(x))$, giving the fourth term. The final equality uses the fact that $s(x) = p(x) + r(x) - p(x)r(x)$. □

Lemma 3. *For any $r \in [0, 1]$ and $p \in [0, 1]$, $h(r, p) \geq 0.003$.*

Proof. Figure 1 contains a (Mathematica-produced) plot of $-h(r, p)$ over the range $r \in [0, 1], p \in [0, 1]$. The plot suggests that $-h(r, p)$ is bounded below zero. The proof of the lemma (which is in the full version of the paper) involves evaluating certain polynomials at about 40 thousand points, and we did this using Mathematica. □

We now have the following theorem.

Theorem 6. *No backoff protocol is positive recurrent when the arrival rate is $\lambda = 0.42$.*

Proof. This follows from Theorem 1, Observation 5 and Lemmas 2 and 3. The value C in Theorem 1 can be taken to be 1 and the value d can be taken to be $1 + B$. □

Now we wish to show that every backoff protocol is transient for $\lambda \geq 0.42$. Once again, fix a backoff protocol $p_0, p_1, \ldots$ with $p_0 = 1$. Notice that our potential function f almost satisfies the conditions in Theorem 2. The main problem is that there is no absolute bound on the amount that f can change in a single step, because the arrivals are drawn from a Poisson distribution. We get around this problem by first considering a *truncated-Poisson* distribution, $T_{M,\lambda}$, in which the

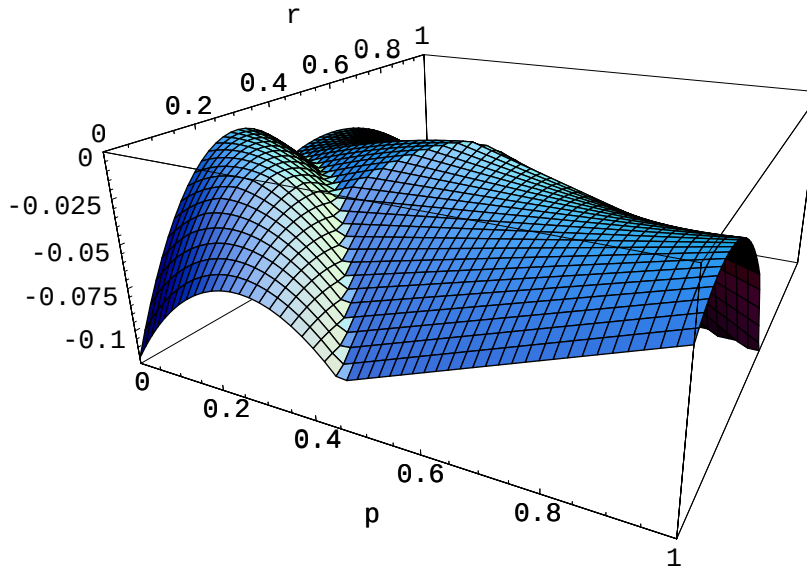


Fig. 1. $-h(r,p)$ over the range $r \in [0,1], p \in [0,1]$.

probability of r inputs is $e^{-\lambda}\lambda^r/r!$ (as for the Poisson distribution) when $r < M$, but $r = M$ for the remaining probability. By choosing M sufficiently large we can have $E[T_{M,\lambda}]$ arbitrarily close to λ. Using methods similar to those used in the proof of Theorem 6 (but using Theorem 2 instead of Theorem 1) we obtain Lemma 4 which in turn (by Observation 4) implies Theorem 7. Lemmas 2 and 3 can also be used (together with deletion resilience (Observation 3)) to show that the *capacity* of every backoff protocol is at most 0.42, so we obtain Theorem 8. For details, see the full version.

Lemma 4. *Every backoff protocol is transient for the input distribution* $T_{M,\lambda}$ *when* $\lambda = 0.42$ *and* $\lambda' = E[T_{M,\lambda}] > \lambda - 0.001$.

Theorem 7. *Every backoff protocol is transient under the Poisson distribution with arrival rate* $\lambda \geq 0.42$.

Theorem 8. *The capacity of every backoff protocol is at most* 0.42.

5 Acknowledgement-Based Protocols

We will prove that every acknowledgement-based protocol is transient for all $\lambda > 0.531$; see Theorem 9 for a precise statement of this claim.

An acknowledgement-based protocol can be viewed a system which, at every step t, decides what subset of the old messages to send. The decision is a probabilistic one dependent on the histories of the messages held. As a technical device for proving our bounds, we introduce the notion of a "genie", which (in general) has more freedom in making these decisions than a protocol.

Since we only consider acknowledgement-based protocols, the behaviour of each new message is independent of the other messages and of the state of the system until after its first send. This is why we ignore new messages until their first send – for Poisson arrivals this is equivalent to the convention that each message sends at its arrival time. As a consequence, we impose the limitation on a genie, that each decision is independent of the number of arrivals at that step.

A *genie* is a random variable over the natural numbers, dependent on the complete history (of arrivals and sends of messages) up to time $t-1$, which gives a natural number representing the number of (old) messages to send at time t. It is clear that for every acknowledgement-based protocol there is a corresponding genie. However there are genies which do not behave like any protocol, e.g., a genie may give a cumulative total number of "sends" up to time t which exceeds the actual number of arrivals up to that time.

We prove a preliminary result for such "unconstrained" genies, but then we impose some constraints reflecting properties of a given protocol in order to prove our final results.

Let $I(t), G(t)$ be the number of arrivals and the genie's send value, respectively, at step t. It is convenient to introduce some indicator variables to express various outcomes at the step under consideration. We use i_0, i_1 for the events of no new arrival, or exactly one arrival, respectively, and g_0, g_1 for the events of no send and exactly one send from the genie. The indicator random variable $S(t)$ for a success at time t is given by $S(t) = i_0 g_1 + i_1 g_0$. Let $\mathrm{In}(t) = \sum_{j \le t} I(j)$ and $\mathrm{Out}(t) = \sum_{j \le t} S(j)$. Define $\mathrm{Backlog}(t) = \mathrm{In}(t) - \mathrm{Out}(t)$. Let $\lambda = \lambda_0 \approx 0.567$ be the (unique) root of $\lambda = e^{-\lambda}$.

Lemma 5. *For any genie and input rate $\lambda > \lambda_0$, there exists $\varepsilon > 0$ such that*

$$\mathrm{Prob}[\mathrm{Backlog}(t) > \varepsilon t \text{ for all } t \ge T] \to 1 \text{ as } T \to \infty.$$

Proof. Let $3\varepsilon = \lambda - e^{-\lambda} > 0$. At any step t, $S(t)$ is a Bernoulli variable with expectation $0, e^{-\lambda}, \lambda e^{-\lambda}$, according as $G(t) > 1, G(t) = 1, G(t) = 0$, respectively, which is dominated by the Bernoulli variable with expectation $e^{-\lambda}$. Therefore $\mathrm{E}[\mathrm{Out}(t)] \le e^{-\lambda} t$, and also, $\mathrm{Prob}[\mathrm{Out}(t) - e^{-\lambda} t < \varepsilon t$ for all $t \ge T] \to 1$ as $T \to \infty$. (To see this note that, by a Chernoff bound, $\mathrm{Prob}[\mathrm{Out}(t) - e^{-\lambda} t \ge \varepsilon t] \le e^{-\delta t}$ for a positive constant δ. Thus,

$$\mathrm{Prob}[\exists t \ge T \text{ such that } \mathrm{Out}(t) - e^{-\lambda} t \ge \varepsilon t] \le \sum_{t \ge T} e^{-\delta t},$$

which goes to 0 as T goes to ∞.) We also have $\mathrm{E}[\mathrm{In}(t)] = \lambda t$ and $\mathrm{Prob}[\lambda t - \mathrm{In}(t) \le \varepsilon t$ for all $t \ge T] \to 1$ as $T \to \infty$, since $\mathrm{In}(t) = \mathrm{Poisson}(\lambda t)$. Since

$$\mathrm{Backlog}(t) = \mathrm{In}(t) - \mathrm{Out}(t) = (\lambda - e^{-\lambda})t + (\mathrm{In}(t) - \lambda t) + (e^{-\lambda} t - \mathrm{Out}(t))$$

$$= \varepsilon t + (\varepsilon t + \mathrm{In}(t) - \lambda t) + (\varepsilon t + e^{-\lambda} t - \mathrm{Out}(t)),$$

the result follows. □

Corollary 2. *No acknowledgement-based protocol is recurrent for $\lambda > \lambda_0$ or has capacity greater than λ_0.*

To strengthen the above result we introduce a restricted class of genies. We think of the messages which have failed exactly once as being contained in *the bucket*. (More generally, we could consider an array of buckets, where the jth bucket contains those messages which have failed exactly j times.) A 1-*bucket genie*, here called simply a *bucket genie*, is a genie which simulates a given protocol for the messages in the bucket and is required to choose a send value which is at least as great as the number of sends from the bucket. For such constrained genies, we can improve the bound of Corollary 2.

For the range of arrival rates we consider, an excellent strategy for a genie is to ensure that at least one message is sent at each step. Of course a bucket genie has to respect the bucket messages and is obliged sometimes to send more than one message (inevitably failing). An *eager* genie always sends at least one message, but otherwise sends the minimum number consistent with its constraints.

An eager bucket genie is easy to analyse, since every arrival is blocked by the genie and enters the bucket. For any acknowledgement-based protocol, let Eager denote the corresponding eager bucket genie. Let $\lambda = \lambda_1 \approx 0.531$ be the (unique) root of $\lambda = (1 + \lambda)e^{-2\lambda}$. The following lemma is proved in the full version.

Lemma 6. *For any eager bucket genie and input rate $\lambda > \lambda_1$, there exists $\varepsilon > 0$ such that*

$$\mathrm{Prob}[\mathrm{Backlog}(t) > \varepsilon t \text{ for all } t \geq T] \to 1 \text{ as } T \to \infty.$$

Let *Any* be an arbitrary bucket genie and let *Eager* be the eager bucket genie based on the same bucket parameters. We may couple the executions of Eager and Any so that the same arrival sequences are presented to each. It will be clear that at any stage the set of messages in Any's bucket is a subset of those in Eager's bucket. We may further couple the behaviour of the common subset of messages. Let $\lambda = \lambda_2 \approx 0.659$ be the (unique) root of $\lambda = 1 - \lambda e^{-\lambda}$.

Lemma 7. *For the coupled genies* Any *and* Eager *defined above, if* Out_A *and* Out_E *are the corresponding output functions, we define* $\Delta\mathrm{Out}(t) = \mathrm{Out}_E(t) - \mathrm{Out}_A(t)$. *For any $\lambda \leq \lambda_2$ and any $\varepsilon > 0$,*

$$\mathrm{Prob}[\Delta\mathrm{Out}(t) \geq -\varepsilon t \text{ for all } t \geq T] \to 1 \text{ as } T \to \infty.$$

Proof. Let c_0, c_1, c_* be indicators for the events of the number of common messages sending being 0, 1, or more than one, respectively. In addition, for the messages which are only in Eager's bucket, we use the similar indicators e_0, e_1, e_*. Let

a_0, a_1 represent Any not sending, or sending, *additional* messages respectively. (Note that Eager's behaviour is fully determined.)

We write $Z(t)$ for $\Delta\mathrm{Out}(t) - \Delta\mathrm{Out}(t-1)$, for $t > 0$, so Z represents the difference in success between Eager and Any in one step. In terms of the indicators we have

$$Z(t) = S_E(t) - S_A(t) = i_0 g_{E1}(t) + i_1 g_{E0}(t) - i_0 g_{A1}(t) - i_1 g_{A0}(t),$$

where $S_E(t)$ is the indicator random variable for a success of Eager at time t and $g_{E1}(t)$ is the event that Eager sends exactly one message during step t (and so on) as in the paragraph before Lemma 5. Thus,

$$Z(t) \geq i_0 c_0 (a_0(e_0 + e_1) - a_1 e_*) - i_0 c_1 (e_1 + e_*) - i_1 c_0 a_0.$$

Note that if the number of arrivals plus the number of common bucket sends is more than 1 then neither genie can succeed. We also need to keep track of the number, ΔB, of extra messages in Eager's bucket. At any step, at most one new such extra message can arrive; the indicator for this event is $i_1 c_0 a_0$, i.e., there is a single arrival and no sends from the common bucket, so if Any does not send then this message succeeds but Eager's send will cause a failure. The number of "extra" messages leaving Eager's bucket at any step is unbounded, given by a random variable we could show as $\mathbf{e} = 1 \cdot e_1 + 2 \cdot e_2 + \cdots$. However $\mathbf{e}$ dominates $e_1 + e_*$ and it is sufficient to use the latter. The change at one step in the number of extra messages satisfies:

$$\Delta B(t) - \Delta B(t-1) = i_1 c_0 a_0 - \mathbf{e} \leq i_1 c_0 a_0 - (e_1 + e_*).$$

Next we define $Y(t) = Z(t) - \alpha(\Delta B(t) - \Delta B(t-1))$, for some positive constant α to be chosen below. Note that $X(t) = \sum_{j=1}^{t} Y(j) = \Delta\mathrm{Out}(t) - \alpha \Delta B(t)$. We also define

$$Y'(t) = i_0 c_0 (a_0(e_0 + e_1) - a_1 e_*) - i_0 c_1 (e_1 + e_*) - i_1 c_0 a_0 - \alpha(i_1 c_0 a_0 - (e_1 + e_*))$$

and $X'(t) = \sum_{j=1}^{t} Y'(j)$. Note that $Y(t) \geq Y'(t)$.

We can identify five (exhaustive) cases A,B,C,D,E depending on the values of the c's, a's and e's, such that in each case $Y'(t)$ dominates a given random variable depending only on $I(t)$.

A. c_*:	$Y'(t) \geq 0$;
B. $(c_1 + c_0 a_1)(e_1 + e_*)$:	$Y'(t) \geq \alpha - i_0$;
C. $(c_1 + c_0 a_1) e_0$:	$Y'(t) \geq 0$;
D. $c_0 a_0 (e_0 + e_1)$:	$Y'(t) \geq i_0 - (1 + \alpha) i_1$;
E. $c_0 a_0 e_*$:	$Y'(t) \geq \alpha - (1 + \alpha) i_1$.

For example, the correct interpretation of Case B is "conditioned on $(c_1 + c_0 a_1)(e_1 + e_*) = 1$, the value of $Y'(t)$ is at least $\alpha - i_0$." Since $\mathrm{E}[i_0] = e^{-\lambda}$ and $\mathrm{E}[i_1] = \lambda e^{-\lambda}$, we have $\mathrm{E}[Y'(t)] \geq 0$ in each case, as long as $\max\{e^{-\lambda}, \lambda e^{-\lambda}/(1 - \lambda e^{-\lambda})\} \leq \alpha \leq 1/\lambda - 1$. There exists such an α for any $\lambda \leq \lambda_2$; for such λ we may take the value $\alpha = e^{-\lambda}$, say.

Let $\mathcal{F}_t$ be the σ-field generated by the first t steps of the coupled process. Let $\hat{Y}(t) = Y'(t) - E[Y'(t) \mid \mathcal{F}_{t-1}]$ and let $\hat{X}(t) = \sum_{i=1}^{t} \hat{Y}(t)$. The sequence $\hat{X}(0), \hat{X}(1), \ldots$ forms a *martingale* (see Definition 4.11 of [20]) since $E[\hat{X}(t) \mid \mathcal{F}_{t-1}] = \hat{X}(t-1)$. Furthermore, there is a positive constant c such that $|\hat{X}(t) - \hat{X}(t-1)| \leq c$. Thus, we can apply the Hoeffding-Azuma Inequality (see Theorem 4.16 of [20]). In particular, we can conclude that

$$\text{Prob}[\hat{X}_t \leq -\epsilon t] \leq 2 \exp\left(-\frac{\epsilon^2 t}{2c^2}\right).$$

Our choice of α above ensured that $E[Y'(t) \mid \mathcal{F}_{t-1}] \geq 0$. Hence $Y'(t) \geq \hat{Y}(t)$ and $X'(t) \geq \hat{X}(t)$. We observed earlier that $X(t) \geq X'(t)$. Thus, $X(t) \geq \hat{X}(t)$ so we have

$$\text{Prob}[X_t \leq -\epsilon t] \leq 2 \exp\left(-\frac{\epsilon^2 t}{2c^2}\right).$$

Since $2 \exp\left(-\frac{\epsilon^2 t}{2c^2}\right)$ converges, we deduce that

$$\text{Prob}[X(t) \geq -\varepsilon t \text{ for all } t \geq T] \to 1 \text{ as } T \to \infty.$$

Since $\Delta\text{Out}(t) = X(t) + \alpha \Delta B(t) \geq X(t)$, for all t, we obtain the required conclusion. □

Finally, we can prove the main results of this section.

Theorem 9. *Let P be an acknowledgement-based protocol. Let $\lambda = \lambda_1 \approx 0.531$ be the (unique) root of $\lambda = (1+\lambda)e^{-2\lambda}$. Then*

1. *P is transient for arrival rates greater than λ_1;*
2. *P has capacity no greater than λ_1.*

Proof. Let λ be the arrival rate, and suppose $\lambda > \lambda_1$. If $\lambda > \lambda_0 \approx 0.567$ then the result follows from Lemma 5. Otherwise, we can assume that $\lambda < \lambda_2 \approx 0.659$. If E is the eager genie derived from P, then the corresponding Backlogs satisfy $\text{Backlog}_P(t) = \text{Backlog}_E(t) + \Delta\text{Out}(t)$. The results of Lemmas 6 and 7 show that, for some $\varepsilon > 0$, both $\text{Prob}[\text{Backlog}_E(t) > 2\varepsilon t \text{ for all } t \geq T]$ and $\text{Prob}[\Delta\text{Out}(t) \geq -\varepsilon t \text{ for all } t \geq T]$ tend to 1 as $T \to \infty$. The conclusion of the theorem follows. □

References

1. D. Aldous, Ultimate instability of exponential back-off protocol for acknowledgement-based transmission control of random access communication channels, *IEEE Trans. Inf. Theory* **IT-33(2)** (1987) 219–233.
2. H. Al-Ammal, L.A. Goldberg and P. MacKenzie, Binary Exponential Backoff is stable for high arrival rates, To appear in *International Symposium on Theoretical Aspects of Computer Science* **17** (2000).

3. J.I. Capetanakis, Tree algorithms for packet broadcast channels, *IEEE Trans. Inf. Theory* **IT-25(5)** (1979) 505-515.
4. A. Ephremides and B. Hajek, Information theory and communication networks: an unconsummated union, *IEEE Trans. Inf. Theory* **44(6)** (1998) 2416–2432.
5. G. Fayolle, V.A. Malyshev, and M.V. Menshikov, *Topics in the Constructive Theory of Countable Markov Chains,* (Cambridge University Press, 1995).
6. L.A. Goldberg and P.D. MacKenzie, Analysis of Practical Backoff Protocols for Contention Resolution with Multiple Servers, *Journal of Computer and Systems Sciences*, **58** (1999) 232–258.
7. L.A. Goldberg, P.D. MacKenzie, M. Paterson and A. Srinivasan, Contention resolution with constant expected delay, Pre-print (1999) available at `http://www.dcs.warwick.ac.uk/~leslie/pub.html`. (Extends a paper by the first two authors in *Proc. of the Symposium on Foundations of Computer Science (IEEE)* 1997 and a paper by the second two authors in *Proc. of the Symposium on Foundations of Computer Science (IEEE)* 1995.)
8. J. Goodman, A.G. Greenberg, N. Madras and P. March, Stability of binary exponential backoff, *J. of the ACM*, **35(3)** (1988) 579–602.
9. A.G. Greenberg, P. Flajolet and R. Ladner, Estimating the multiplicities of conflicts to speed their resolution in multiple access channels, *J. of the ACM*, **34(2)** (1987) 289–325.
10. G.R. Grimmet and D.R. Stirzaker, *Probability and Random Processes, Second Edition.* (Oxford University Press, 1992)
11. J. Håstad, T. Leighton and B. Rogoff, Analysis of backoff protocols for multiple access channels, *SIAM Journal on Computing* **25(4)** (1996) 740-774.
12. F.P. Kelly, Stochastic models of computer communication systems, *J.R. Statist. Soc. B* **47(3)** (1985) 379–395.
13. F.P. Kelly and I.M. MacPhee, The number of packets transmitted by collision detect random access schemes, *The Annals of Probability,* **15(4)** (1987) 1557–1568.
14. U. Loher, Efficiency of first-come first-served algorithms, *Proc. ISIT* (1998) p108.
15. U. Loher, Information-theoretic and genie-aided analyses of random-access algorithms, PhD Thesis, Swiss Federal Institute of Technology, DISS ETH No. 12627, Zurich (1998).
16. I.M. MacPhee, On optimal strategies in stochastic decision processes, D. Phil Thesis, University of Cambridge, (1987).
17. R.M. Metcalfe and D.R. Boggs, Ethernet: Distributed packet switching for local computer networks. *Commun. ACM,* **19** (1976) 395–404.
18. M. Molle and G.C. Polyzos, Conflict resolution algorithms and their performance analysis, Technical Report CS93-300, Computer Systems Research Institute, University of Toronto, (1993).
19. J. Mosely and P.A. Humblet, A class of efficient contention resolution algorithms for multiple access channels, *IEEE Trans. on Communications,* **COM-33(2)** (1985) 145–151.
20. R. Motwani and P. Raghavan, *Randomized Algorithms* (Cambridge University Press, 1995.)
21. P. Raghavan and E. Upfal, Contention resolution with bounded delay, *Proc. of the ACM Symposium on the Theory of Computing* **24** (1995) 229–237.
22. R. Rom and M. Sidi, Multiple access protocols: performance and analysis, (Springer-Verlag, 1990).
23. B.S. Tsybakov and N. B. Likhanov, Upper bound on the capacity of a random multiple-access system, *Problemy Peredachi Informatsii,* **23(3)** (1987) 64–78.

24. B.S. Tsybakov and V. A. Mikhailov, Free synchronous packet access in a broadcast channel with feedback, *Probl. Information Transmission,* **14(4)** (1978) 259–280.
25. S. Verdu, Computation of the efficiency of the Mosely-Humblet contention resolution algorithm: a simple method, *Proc. of the IEEE* **74(4)** (1986) 613–614.
26. N.S. Vvedenskaya and M.S. Pinsker, Nonoptimality of the part-and-try algorithm, *Abstr. Papers, Int. Workshop "Convolutional Codes; Multi-User Commun.,"* Sochi, U.S.S.R, May 30 - June 6, 1983, 141–144.

Deterministic Radio Broadcasting

Bogdan S. Chlebus[1], Leszek Gąsieniec[2],
Anna Östlin[3], and John Michael Robson[4]

[1] Instytut Informatyki, Uniwersytet Warszawski, Banacha 2, 02-097 Warszawa, Poland. chlebus@mimuw.edu.pl
[2] Department of Computer Science, The University of Liverpool, Liverpool L69 7ZF, United Kingdom. leszek@csc.liv.ac.uk
[3] Department of Computer Science, Lund University, Box 118, S-221 00 Lund, Sweden. Anna.Ostlin@cs.lth.se
[4] LaBRI, Université Bordeaux 1, 351, cours de la Libération, 33405 Talence, France. robson@labri.u-bordeaux.fr

Abstract. We consider broadcasting in radio networks: one node of the network knows a message that needs to be learned by all the remaining nodes. We seek distributed deterministic algorithms to perform this task. Radio networks are modeled as directed graphs. They are unknown, in the sense that nodes are not assumed to know their neighbors, nor the size of the network, they are aware only of their individual identifying numbers. If more than one message is delivered to a node in a step then the node cannot hear any of them. Nodes cannot distinguish between such collisions and the case when no messages have been delivered in a step.

The fastest previously known deterministic algorithm for deterministic distributed broadcasting in unknown radio networks was presented in [6], it worked in time $\mathcal{O}(n^{11/6})$. We develop three new deterministic distributed algorithms. Algorithm A develops further the ideas of [6] and operates in time $\mathcal{O}(n^{1.77291}) = \mathcal{O}(n^{9/5})$, for general networks, and in time $\mathcal{O}(n^{1+a+\mathrm{H}(a)+o(1)})$ for sparse networks with in-degrees $\mathcal{O}(n^a)$ for $a < 1/2$; here H is the entropy function. Algorithm B uses a new approach and works in time $\mathcal{O}(n^{3/2}\log^{1/2} n)$ for general networks or $\mathcal{O}(n^{1+a+o(1)})$ for sparse networks. Algorithm C further improves the performance for general networks running in time $\mathcal{O}(n^{3/2})$.

Keywords: Broadcasting, Distributed, Deterministic, Radio network

The work of the first, third and fourth authors has been partly done while visiting the University of Liverpool. The work of the first author was partly supported by EPSRC grant GR/M75105. The work of the second author was partly supported by EPSRC grant GR/M75105 and Nuffield Foundation award for newly appointed lecturers NUFNAL. The work of the third author has been supported in part by The Royal Swedish Academy of Science. The work of the fourth author has been supported by the British French Alliance program.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 717–729, 2000.
© Springer-Verlag Berlin Heidelberg 2000

1 Introduction

Wireless communication has become popular due to the recent advent of new technologies. It is present not only in the ubiquitous cordless and cellular phones, but also in personal communication services, mobile data networks, and local-area wireless networks (cf. [15]). Protocols used to communicate over wireless networks assume as little as possible about the topology of the network, because it may change over time, especially in land-mobile radio networks. Communication over such networks creates challenging algorithmic problems.

We consider an abstraction of a wireless network called a *radio network* and modeled as a directed graph. An edge from node v_1 to v_2 corresponds to the fact that messages transmitted by v_1 can be directly received by v_2; in other words, v_2 is in the range of the transmitter at v_1. The underlying feature of communication in radio networks is that a node that simultaneously receives messages from at least two transmitters cannot hear any of them because of mutual interference. If a node cannot hear a message then it hears some noise, distinct from any meaningful message. If the noise heard by a node while no messages have been sent to it is distinct from the noise heard when many messages have been sent to it then the model is said to be with collision detection.

We consider the problem of dissemination of information in radio networks. In its simplest variant, we have one node of the network, called a *source*, which stores a message that needs to be learned by all the remaining nodes. This specific communication task is called the problem of *broadcasting*. We seek distributed (decentralized) algorithms to perform broadcasting in radio networks. We restrict our attention to deterministic algorithms.

The nodes of the network do not know the topology of the underlying graph, except for their own individual numbers and possibly the total number of nodes. In particular, a node is not assumed to know the numbers of the nodes with whom it could communicate directly.

The nodes of the network have access to a global clock and operate in steps. The performance of algorithms is measured by their worst-case time behavior over all the possible networks of a given size.

Review of prior work. A lower bound $\Omega(n)$ for deterministic distributed broadcasting in unknown radio networks was proved in [4]. The best currently known lower bound is $\Omega(n \lg n)$, see [6]. The first distributed deterministic algorithms for unknown radio networks were presented in [8]; however the networks considered there were quite restricted, namely, nodes were assumed to be located in a line, and each node could reach directly all the nodes within a certain distance. A systematic study of deterministic distributed algorithms in unknown radio networks modeled as directed graphs was undertaken in [6]. This paper compared the broadcasting power of radio networks distinguished by the availability of collision detection. The problem of broadcasting was considered in [6] in two variants, depending on whether the source was required to be informed about the task having been completed (*radio broadcasting with acknowledgement*) or not (*radio broadcasting without acknowledgement*). It was shown that the former task could not be performed on a radio network if nodes do not know

the size of the network in the model without collision detection. This was shown to hold even if the underlying graph is symmetric, that is, for each edge in the graph the edge with the reversed direction is also available. Algorithms were developed in [6] for the problem of broadcasting without acknowledgement in the model without collision detection. One of them operated in time $\mathcal{O}(n)$, but was restricted to the case when the underlying graph was symmetric. The algorithm developed in [6] for general networks had performance $\mathcal{O}(n^{11/6})$. The model with collision detection is capable of performing acknowledged radio broadcasting, it was shown in [6] how to achieve this in time $\mathcal{O}(n)$ in symmetric graphs, and in time $\mathcal{O}(n \cdot \mathrm{ecc})$ for general strongly connected graphs, where ecc is the largest distance from the source to any other node.

Summary of contributions. We consider the problem of broadcasting in unknown radio networks by distributed algorithms. The model is without collision detection. The broadcasting is without acknowledgement. We develop three deterministic algorithms.

Algorithm A operates in time $\mathcal{O}(n^{2-\lambda+\epsilon})$, for any $\epsilon > 0$, for general networks. The constant λ is the root of the equation $\lambda + \mathrm{H}(\lambda) = 1$, where H is the (binary) entropy function; numerically, the bound is $\mathcal{O}(n^{1.77291}) = \mathcal{O}(n^{9/5})$. Algorithm A operates in time $\mathcal{O}(n^{1+a+\mathrm{H}(a)+o(1)})$ for sparse networks, in which in-degrees of nodes are $\mathcal{O}(n^a)$ for $a < 1/2$. Algorithm A is developed by employing many selective families of subsets of $[1..n]$ simultaneously. The notion of a selective family was introduced in [6] to develop an algorithm working in time $\mathcal{O}(n^{11/6})$, which used just two special selective families.

Algorithm B uses transmissions based on the processor ID modulo p for many primes p. It runs in time $\mathcal{O}(n^{3/2} \log^{1/2} n)$ or $\mathcal{O}(n^{1+a+o(1)})$ for sparse networks.

Algorithm C works in time $\mathcal{O}(n^{3/2})$. The underlying paradigm is to take a prime number p such that $p^2 \geq n$ and to use for simultaneous transmission sets of points in $[0..p-1] \times [0..p-1]$ that are lines described by equations over the field $\mathbb{F}_p$. Algorithm C is the fastest that we know of in the general case.

Other related work. Early work on radio communication dealt with the *single-hop radio network* model, which is a channel accessible by all the processors (see [3,10]), for recent results see [11]. This is a special case of our model when the graph is complete and symmetric. The general case has been called the *multi-hop radio network*.

Much of the previous work on distributed broadcasting in radio networks has concentrated on randomized algorithms ([3,4,5]). In [4] a randomized protocol was developed that works in time $\mathcal{O}((D + \lg n/\epsilon) \cdot \lg n)$ with probability $1 - \epsilon$, where D is the diameter of the network. This is close to optimal as follows from the known lower bounds. One of them, proved in [1], is $\Omega(\lg^2 n)$, and holds even for graphs of a constant depth; the other $\Omega(D \lg(n/D))$ was shown in [13].

Another area of research on radio networks was how to compute an optimal schedule of broadcasting, by a centralized algorithm, given a description of the network. It was shown in [7] that this problem is NP-hard; this holds true even when restricted to graphs induced by nodes located in the plane and edges

determined by ranges of nodes, see [16]. On the other hand, it was shown in [9] that broadcasting can be performed in time $\mathcal{O}(D + \lg^5 n)$.

Simulations between the synchronous point-to-point message-passing model and the radio model were presented in [2]. Fault-tolerance issues of distributed radio broadcasting were studied in [12,14].

2 Model of Computation

Radio network. The network is modeled as a directed graph. Node v_1 is a *neighbor* or *successor of node* v_2 if there is a link from v_2 to v_1. It is assumed that for any node v of the network there is a directed path from the source to v.

In each step a node may choose to be in one of two modes: either the *receive mode* or the *broadcast mode*. A node in the broadcast mode transmits a message along all its out-going links. A message is delivered to all the recipients in the step in which it was sent. A node in the receive mode attempts to hear the messages delivered along all its in-coming links. If a message has been delivered to a node it does not necessarily mean that the node is able to hear it. The basic feature of the radio network is that if more than one link brings in messages to a node v during a step then v does not hear any of the messages, and all of them are lost as far as v is concerned. Node v can hear a message delivered along a link if this link is the only one bringing in a message. If more than one message arrives at a node at one step then a *collision* at the node is said to occur. The model of radio communication is in two variants, depending on the ability of nodes to detect collisions. If a node cannot hear a message during a step then it may be assumed to hear some default signal called *noise*. Noise is distinct from any meaningful source messages. If there is only one noise signal then it is called the *background noise* and the model is said to be *without collision detection*. The model *with collision detection* allows two different noise signals: no messages delivered produces the background noise, but more than one produce *interference noise*. In this paper we work with the model without collision detection.

Local knowledge. If the network consists of n nodes then each of them is assigned a unique integer in the range 0 through $n-1$ which is the identification number (ID) of the node. Nodes have a restricted knowledge of the underlying graph. Each node knows only its ID, and if it is the source node. Nodes are not assumed to know the IDs of nodes to which they are connected. Nodes are also not assumed to know the size of the network; however, to simplify the exposition of algorithms we refer explicitly to the size n; this can be avoided, see Section 6.

A broadcasting algorithm is said to have *completed broadcasting* when all the nodes have learned the source message. When this happens the nodes need not be aware of it. Nodes do not send any feedback or acknowledgement information to other nodes. The algorithm we present may terminate after a prescribed number of steps, which is yielded by the performance bounds that we prove.

Distributed setting. The communication protocol operates in steps synchronized by a global clock. During a step each node may make an attempt to send a message or to receive a message. The nodes of the network are processing units,

each able to perform local sequential computations. Whenever a node needs to perform local computation to decide the next send/receive operation, it is assumed that this local computation can always be performed during the current step. Similarly we do not assume any restrictions on the size of local memories.

At each step a node decides whether to broadcast, to remain quiet or to terminate. The broadcast mode is entered by a node only if the source message has been already received by that node. Each decision of node v concerning broadcast/receive mode at a current step depends on the following information:

- the status of v: is v the source node or not;
- the number of the current step;
- the size n of the network;
- the ID of node v;
- if v has already heard the source message or not.

Algorithms. An algorithm is represented by a sequence of sets $T_0, T_1, T_2, \ldots$ called *transmissions*. Each $T_i \subseteq [0..n-1]$ is a set of IDs of nodes. Set T_0 consists of only the source node. A node v enters the broadcast mode at step i, and hence sends the source message to its neighbors, if and only if the following conditions are simultaneously satisfied:

- the ID of v is in T_i;
- node v has already heard the source message.

A simple algorithm was defined in [6] by taking $T_i = \{i \bmod n\}$, for $i > 0$. We call it the *round-robin algorithm*. It completes broadcasting in time $\mathcal{O}(n^2)$

(Notation lg denotes the logarithm to the base 2. We assume that each number in $[0..n-1]$ has exactly $\lfloor \lg n \rfloor + 1$ digits in its binary representation, obtained by padding the most significant positions with zeros if necessary. The least significant position has number 1 and the most significant has number $\lfloor \lg n \rfloor + 1$.)

Broadcasting paradigm. A node evolves through three conceptual states in the course of a broadcasting algorithm. If it does not know the source message yet, it is *uninformed*. A node becomes *active* in the step when it learns the source message for the first time. In the beginning the source node is the only active node and the remaining nodes are all uninformed. An active node changes its status to *passive* when the last of its neighbors learns the source message.

Lemma 1. *If a node broadcasts as the only active node it becomes passive.*

Proof. Since uninformed nodes do not broadcast, the active node i and possibly certain passive nodes are the only ones which broadcast. Passive nodes cannot interfere with the receipt of the message by uninformed neighbours of i since passive nodes have no uninformed neighbours.

Each time a node changes its status we say that *progress* has been made. Each change from uninformed to active or from active to passive contributes a unit to the measure of progress, and change from uninformed to passive contributes two units. After progress $2n-1$ has been accrued we know that all the nodes are passive, so all the nodes know the source message by the definition of passive.

Selective families. Following [6], we say that a family $\mathcal{R}$ of subsets of $[0..n-1]$ is *y-selective*, for a positive integer y, if for any subset $Z \subseteq [0..n-1]$ such that $|Z| \leq y$ there is a set $S \in \mathcal{R}$ such that $|S \cap Z| = 1$. We also say that a family $\mathcal{R}$ of subsets of $[0..n-1]$ is *strongly y-selective*, for any positive integer y, if for any subset $Z \subseteq [0..n-1]$ such that $|Z| \leq y$, and any $z \in Z$, there is a set $S \in \mathcal{R}$ such that $S \cap Z = \{z\}$.

3 Selective Families Streamlined

In [6] specific selective families were considered defined as follows. Let $W = \langle w_1, \ldots, w_k \rangle$ be a sequence of integers such that $1 \leq w_1 < w_2 < \cdots < w_k \leq \lfloor \lg n \rfloor + 1$. Let $B = \langle b_1, \ldots, b_k \rangle$ be a sequence of bits, each equal to either 0 or 1. Set $S(W, B)$ is defined to consist exactly of those integers in $[0..n-1]$ which have the w_i-th digit in their binary expansions equal to b_i $(1 \leq i \leq k)$.

The number k used above is said to be the *width* of sets W, B and $S(W, B)$. The family $\mathcal{R}(k)$ is defined to consist of all the sets of width k of the form $S(W, B)$. Hence set $S = S(W, B) \in \mathcal{R}(k)$ is determined by selecting both a certain k positions in the binary expansions of numbers in $[0..n-1]$ and also the digits on those positions.

The key property of $\mathcal{R}(k)$ is that it is 2^k-selective ([6]). To see this, take $Z \subseteq [0..n-1]$ where $|Z| \leq 2^k$. Let w_1 be the smallest integer such that there are two elements in Z such that their binary expansions differ on position w_1. This means that the sets Z_i consisting of those $x \in Z$ that the w_1-th digit in the binary expansion of x is i, for $i \in \{0, 1\}$, are both nonempty and disjoint. Let b_1 be the number with the property that $|Z_{b_1}| \leq |Z_{1-b_1}|$. Hence $0 < |Z_{b_1}| \leq \frac{1}{2}|Z|$. Replacing Z by Z_{b_1} and continuing in a similar fashion we obtain sequences $W = \langle w_1, \ldots, w_i \rangle$ and $B = \langle b_1, \ldots, b_i \rangle$ such that $|S(W, B) \cap Z| = 1$ and $i \leq k$. They can be padded to sequences of length exactly k if $i < k$.

Sets from families $\mathcal{R}(k)$ can be used as transmissions. The round-robin algorithm uses just singleton sets, which is the family $\mathcal{R}(\lfloor \lg n \rfloor + 1)$. The algorithm of [6] uses these singletons interleaved with some other specific family $\mathcal{R}(k)$.

We use $\mathcal{O}(\lg n)$ families $\mathcal{R}(k)$ simultaneously. Algorithm A is organized as a loop which iterates phases. A *phase* consists of $\mathcal{O}(\lg n)$ transmissions, each from a different family $\mathcal{R}(k)$. The algorithm cycles through all the elements in $\mathcal{R}(k)$. To be specific, let us fix an ordering $\langle S_1, S_2, \ldots \rangle$ of all the sets in $\mathcal{R}(k)$. Then in phase i set S_j is the transmission from $\mathcal{R}(k)$, where $j = i \bmod |\mathcal{R}(k)|$.

Family $\mathcal{R}(\lfloor \lg n \rfloor + 1)$ has exactly n singleton sets. So during n consecutive phases each node with the source message will have an opportunity to be the only node performing broadcasting in the whole network. Hence its neighbors will all hear the message. We do not expect to gain much using sets from $\mathcal{R}(k)$ as transmissions if $|\mathcal{R}(k)| > n$, and they do not matter as far as the analysis of performance of algorithm A is concerned.

Let $k = x \cdot \lg n$, for $0 < x \leq 1/2$. Then

$$|\mathcal{R}(k)| = 2^k \binom{\lfloor \lg n \rfloor + 1}{k} = 2^{x \lg n} \binom{\lfloor \lg n \rfloor + 1}{x \lg n} = n^x n^{\mathrm{H}(x) + o(1)} = n^{x + \mathrm{H}(x)} n^{o(1)}$$

where $\mathrm{H}(x) = -[x \lg x + (1-x)\lg(1-x)]$ is the (binary) entropy function.

Let $0 < \lambda < 1/2$ be the solution of the equation $\lambda + \mathrm{H}(\lambda) = 1$, $\lambda = 0.22709....$ The families $\mathcal{R}(k)$ for $k \le \lambda \lg n$ have $\mathcal{O}(n)$ subsets of $[0..n-1]$ each.

Algorithm A has phases that use transmissions from each $\mathcal{R}(k)$ such that $k \le \lambda \lg n$, and additionally from $\mathcal{R}(\lfloor \lg n \rfloor + 1)$, which are singletons.

Let us consider the contribution to progress from any $s+1$ families $\mathcal{R}(k_1)$, $\mathcal{R}(k_2)$, $\ldots$, $\mathcal{R}(k_s)$ and $\mathcal{R}(\lfloor \lg n \rfloor + 1)$, where $k_1 < k_2 < \cdots < k_s < \lfloor \lg n \rfloor + 1 = k_{s+1}$. We assume $|\mathcal{R}(k_i)| = \mathcal{O}(n)$. Transmissions from these families are interleaved, during step t we use a transmission from $\mathcal{R}(k_x)$ where $x = (t \bmod (s+1)) + 1$. Transmissions from $\mathcal{R}(k_i)$ are used in some cyclical order.

Let F be the set of active nodes at the start of a specific phase t. We consider a number of cases.

Case 1: A progress is made during the phases t through $t + |\mathcal{R}(k_1)|$.

If Case 1 does not hold, consider the process described above demonstrating that $\mathcal{R}(k_1)$ is 2^{k_1}-selective but now take both $b_j = 0$ and $b_j = 1$ at step j; this will give 2^{k_1} transmissions $C_1 \subseteq \mathcal{R}(k_1)$ such that all the sets of the form $x \cap F$, for $x \in C_1$, are disjoint, and each includes at least two elements.

If $T_1 \in \mathcal{R}(x)$, $T_2 \in \mathcal{R}(y)$, $x < y$ and $T_2 \subset T_1$ then T_2 is said to *refine* T_1.

Case 2: Case 1 does not hold and during the phases t through $t + |\mathcal{R}(k_2)|$ at least half of transmissions in C_1 are refined by transmissions in $\mathcal{R}(k_2)$, each of them including exactly one element from F.

If neither of Cases 1 and 2 holds then there is a set $C_1' \subseteq C_1$ of transmissions such that $|C_1'| = \frac{1}{2}|C_1|$, and each transmission $T \in C_1'$ is refined by $2^{k_2 - k_1}$ transmissions in some $A(T) \subseteq \mathcal{R}(k_2)$ such that if $S \in A(T)$ then $|S \cap F| > 1$. Let $C_2 = \bigcup_{T \in C_1'} A(T)$. Then $|C_2| = \frac{1}{2} \cdot 2^{k_2}$.

Case 3: Cases 1 or 2 do not hold and during the phases t through $t + |\mathcal{R}(k_3)|$ at least half of the transmissions from C_2 are refined by transmissions from $\mathcal{R}(k_3)$, each including exactly one element from F.

Cases up to s are defined similarly by properties of sets C_i, for $1 \le i < s$. If Case s holds then during the phases t through $t + |\mathcal{R}(k_s)|$ at least half of the transmissions from C_{s-1} are refined by transmissions from $\mathcal{R}(k_s)$ such that each of them includes exactly one element from F.

Case $s+1$: None of the Cases 1 through s holds.

If Case $s+1$ holds then $|F| \ge 2^{-s} \cdot 2^{k_s}$.

Let us estimate the average progress per phase over a period starting at phase t when Case i holds; it is denoted by α_i. If Case 1 holds then

$$\alpha_1 \ge |\mathcal{R}(k_1)|^{-1}$$

during $|\mathcal{R}(k_1)|$ phases. If Case i holds, for $2 \le i \le s$, then

$$\alpha_i \ge 2^{k_{i-1}} \cdot 2^{-i+1} \cdot |\mathcal{R}(k_i)|^{-1}$$

during $|\mathcal{R}(k_i)|$ phases. If Case $s+1$ holds then during n phases the round robin contributes progress at least

$$\alpha_{s+1} \geq |F| \cdot n^{-1} \geq 2^{k_s} \cdot 2^{-s} \cdot n^{-1} \ .$$

We are guaranteed an average progress among $\alpha_1, \ldots, \alpha_{s+1}$, because one of the cases 1 through $s+1$ always holds. To make the estimations of these numbers largest we seek to make our estimates of $\alpha_1, \ldots, \alpha_{s+1}$ asymptotically equal:

$$|\mathcal{R}(k_1)|^{-1} = \Theta(2^{k_1} \cdot |\mathcal{R}(k_2)|^{-1}) = \cdots = \Theta(2^{k_{s-1}} |\mathcal{R}(k_s)|^{-1}) = \Theta(2^{k_s} \cdot n^{-1}) \ . \tag{1}$$

Let $k_i = a_i \lg n$. Equations (1) translate into

$$-a_1 - \mathrm{H}(a_1) = a_1 - a_2 - \mathrm{H}(a_2) = \cdots = a_{s-1} - a_s - \mathrm{H}(a_s) = a_s - 1 \ . \tag{2}$$

Equations (2) are equivalent to the following system of equations:

$$\begin{aligned} 2a_s + \mathrm{H}(a_s) &= a_{s-1} + 1 \\ 2a_{s-1} + \mathrm{H}(a_{s-1}) &= a_{s-2} + a_s + \mathrm{H}(a_s) \\ &\cdots \\ 2a_2 + \mathrm{H}(a_2) &= a_1 + a_3 + \mathrm{H}(a_3) \\ 2a_1 + \mathrm{H}(a_1) &= a_2 + \mathrm{H}(a_2) \end{aligned}$$

Let function f be defined as $f(x) = x + \mathrm{H}(x)$, for $0 \leq x \leq 1/2$. Our system of equations can be rewritten as follows using notation with f:

$$\begin{aligned} 1 - f(a_s) &= a_s - a_{s-1} \\ a_{s-1} - a_{s-2} &= f(a_s) - f(a_{s-1}) \\ a_{s-2} - a_{s-3} &= f(a_{s-1}) - f(a_{s-2}) \\ &\cdots \\ a_2 - a_1 &= f(a_3) - f(a_2) \\ a_1 &= f(a_2) - f(a_1) \end{aligned}$$

We can equate $f(y) - f(x) = f'(z)(y - x)$ for certain z satisfying $x \leq z \leq y$. Numbers $a_1, \ldots, a_s$ are in $[0..\lambda]$, where $f(\lambda) = 1$. If $0 < z < \lambda$ then $f'(z) > f'(\lambda) > f'(1/2) = 1$. We obtain the following estimation:

$$\begin{aligned} 1 - f(a_s) = a_s - a_{s-1} &\leq \frac{1}{f'(\lambda)}(f(a_s) - f(a_{s-1})) \\ = \frac{1}{f'(\lambda)}(a_{s-1} - a_{s-2}) &\leq \frac{1}{[f'(\lambda)]^2}(f(a_{s-1}) - f(a_{s-2})) \leq \ldots \\ \leq \frac{1}{[f'(\lambda)]^{s-1}}(f(a_2) - f(a_1)) &\leq a_1 \cdot [f'(\lambda)]^{1-s} \leq [f'(\lambda)]^{1-s} \ . \end{aligned}$$

This shows that $1 - f(a_s)$ converges to 0 as $s \to \infty$. It follows that $\lim_{s\to\infty} a_s = \lambda$ because f is a continuous function.

Theorem 1. *For any $\epsilon > 0$, algorithm* A *completes broadcasting in time $\mathcal{O}(n^{2-\lambda+\epsilon})$, for $\lambda = 0.22709...$ defined by $\lambda + \mathrm{H}(\lambda) = 1$.*

Proof. The average progress after phase t is at least

$$2^{k_s} \cdot n^{-1} = 2^{a_s \lg n} \cdot n^{-1} = n^{a_s - 1}$$

per phase. To achieve progress $2n-1$ we need $\mathcal{O}(n/n^{a_s-1}) = \mathcal{O}(n^{2-a_s})$ phases. A phase takes $\mathcal{O}(\lg n) = \mathcal{O}(n^{\epsilon})$ steps, and $\lim_{s\to\infty} a_s = \lambda$.

Corollary 1. *Algorithm* A *operates in time $\mathcal{O}(n^{1.77291}) = \mathcal{O}(n^{9/5})$.*

Next we consider graphs with bound $d = d(n)$ on in-degrees of nodes.

Theorem 2. *Algorithm* A *completes broadcasting in time $\mathcal{O}(n \cdot \binom{\lg n}{\lceil \lg d \rceil} \cdot d \lg n)$ if in-degrees are bounded by d.*

Proof. Consider the contribution to progress made by $\mathcal{R}(k)$ for $k = \lceil \lg d \rceil$. Let v be an uninformed node that is a neighbor of an active node. There are at most d such active nodes for v. Family $\mathcal{R}(k)$ is d-selective. Hence during $\mathcal{O}(|\mathcal{R}(k)|)$ phases there will be a step where just one of these active nodes broadcasts and informs node v. The total time of algorithm A is thus bounded by $\mathcal{O}(n|\mathcal{R}(k)| \lg n) = \mathcal{O}(n \cdot \binom{\lg n}{\lceil \lg d \rceil} \cdot d \lg n)$.

Corollary 2. *If $d = \mathcal{O}(n^a)$, for $a < 1/2$, then algorithm* A *completes broadcasting in time $\mathcal{O}(n^{1+a+\mathrm{H}(a)+\epsilon})$, for any constant $\epsilon > 0$.*

4 Multi-prime Algorithms

Let p_i denote the i-th prime number and $k(x)$ denote the smallest integer such that $\prod_{i=1}^{k(x)} p_i > n^{x-1}$. Every transmission will be of the form $S_{i,j} = \{m | 0 \le m \le n-1 \text{ and } m = j \pmod{p_i}\}$ for some prime p_i and some $j < p_i$. The set of all transmissions $S_{i,j}$ for a prime p_i is called a p_i *stage* and two stages will be called *different* if they use different primes p_i and p_j. The union of all p_i stages with $p_i \le k(2^s)$ will be called the *s-family*. We note that it is a strongly (2^s)-selective family since every node transmits once for each prime p_i and cannot be blocked by a given other active node for a set of primes with product greater than n.

Lemma 2. *If at time t_0 some uninformed node m has l informed predecessors and no uninformed node has more than l informed predecessors and the transmissions between t_0 and t_1 include different p_{i_j} stages ($1 \le j \le m$) such that $\prod_{j=1}^{m} p_{i_j} \ge n^{2l-1}$, then there is progress at least l between t_0 and t_1.*

Proof. Let $m'_1, \cdots, m'_l$ be the informed predecessors of m. If they all become passive or l new nodes become active between t_0 and t_1, then there has been progress at least l. Otherwise some m'_k remains active so it has a successor m^*

which remains uninformed; m^* had at most l informed predecessors at t_0 and acquired less than l more between t_0 and t_1. The union of the p_{i_j} stages is a strongly $(2l)$-selective family so there is a transmission between t_0 and t_1 in which m'_k and none of these other predecessors of m^* transmits. So m^* would become active at the latest at that transmission so that this case cannot arise.

We can build different versions of our algorithm using $s+1$-families depending on whether a bound D on the in-degree is known and whether we want an algorithm which takes advantage of the existence of low in-degree.
Algorithm B1(D) repeats the s-family (ordered as a succession of its $k(2^s)$ stages) for $s = \lceil \lg 2D \rceil$.
Algorithm B2(D) interleaves $B1(2^s)$ for all s up to $\lceil \lg 2D \rceil$.
Algorithm B3 interleaves round robin with $B1(\sqrt{n/\lg n})$.
Algorithm B4 interleaves B3 with $B2(\sqrt{n}/\lg^{3/2} n)$.

Theorem 3. *Algorithms B run in time:*
B1: $\mathcal{O}(nD \lg^2 n/\lg(D \lg n))$ *if all in-degreess are at most* D*;*
B2: $\mathcal{O}(nd \lg D \lg^2 n/\lg(d \lg n))$ *if all in-degrees are at most* $d \leq D$*;*
B3: $\mathcal{O}(n^{3/2} \lg^{1/2} n)$*;*
B4: $\mathcal{O}(min(n^{3/2} \lg^{1/2} n,\ nd \lg^3 n/\lg(d \lg n)))$ *if all in-degrees are at most* d*.*

Proof.
B1: Since no uninformed node can have more than D predecessors, using Lemma 2 we can divide the time into intervals where each interval achieves progress at least l in a sequence of stages using primes with product less than n^{2l}. Since these primes are $\mathcal{O}(D \lg n)$, their sum is $\mathcal{O}(\frac{\lg n^{2l}}{\lg(D \lg n)} D \lg n) = \mathcal{O}(\frac{lD \lg^2 n}{\lg(D \lg n)})$. This sum is the number of steps in the interval, giving time per unit progress of $\mathcal{O}(\frac{D \lg^2 n}{\lg(D \lg n)})$.
B2: B1(d) would terminate in time $\mathcal{O}(nd \lg^2 n/\lg(d \lg n))$ on its own and so will terminate in time $\mathcal{O}(nd \lg D \lg^2 n/\lg(d \lg n))$ when running interleaved with $\mathcal{O}(\lg D)$ other algorithms.
B3: Starting at any time when the number of active processors is less than $\sqrt{n/\lg n}$, Lemma 2 gives us an interval with time per unit progress of $\mathcal{O}(\sqrt{n \lg n})$ by plugging in $D = \sqrt{n/\lg n}$ in B1; if the number of active processors is at least $\sqrt{n/\lg n}$, the round robin gives progress at least $\sqrt{n/\lg n}$ in the succeeding interval of length $2n$. In both cases we obtain an interval with time per unit progress of $\mathcal{O}(\sqrt{n \lg n})$.
B4: If $d < \sqrt{n/\lg^{3/2} n}$, the interleaved $B2$ guarantees termination within $\mathcal{O}(nd \lg^3 n/\lg(d \lg n))$ steps; otherwise the $B3$ guarantees termination within $\mathcal{O}(n^{3/2} \lg^{1/2} n)$ steps. In each case this gives the time bound stated.

5 Single-Prime Algorithm

Let p be the smallest prime greater than or equal to $\lfloor \sqrt{n} \rfloor$. Then $p^2 = \mathcal{O}(n)$.

Each of the numbers $i \in [0..p^2 - 1]$ can be represented uniquely by a pair of integers $\langle x, y\rangle$ where $i = x \cdot p + y$ and $0 \leq x, y \leq p - 1$. Such a pair $\langle x, y\rangle$ are the *coordinates* of i. We treat all the numbers in $[0..p^2 - 1]$ as IDs of nodes. We refer to them by their coordinates rather than IDs.

The integers in [0..p-1] form a field $\mathbb{F}_p$ if the arithmetic is modulo prime p. For $a, b, c \in \mathbb{F}_p$, such that at least one among the numbers a and b is distinct from 0, let $L(a, b, c)$ be the set of nodes whose coordinates $\langle x, y\rangle$ satisfy the equation

$$a \cdot x + b \cdot y = c$$

in $\mathbb{F}_p$. Each set of nodes of the form $L(a, b, c)$ is called a *line*. Each line has size p. There are exactly $p - 1$ lines disjoint with a given one. Two disjoint lines are said to be *of the same direction*. The total number of lines is $p(p + 1) = \mathcal{O}(n)$. Each node belongs to $p + 1$ lines. For any two different nodes v_1 and v_2 there is exactly one line containing v_1 and v_2. For any two lines L_1 and L_2 of different directions there is exactly one node v such that v belongs to both L_1 and L_2.

Algorithm C uses singleton sets and lines, singletons in even-numbered steps and lines in odd-numbered ones. During each consecutive p^2 even-numbered steps each singleton is used exactly once. Similarly, during each consecutive $p(p + 1)$ odd-numbered steps each line is used exactly once. Moreover, lines of the same direction are used in consecutive odd-numbered steps. A *stage* consists of $2(p+1)$ consecutive steps during which all the lines of the same direction are used.

Lemma 3. *Let F be a set of nodes such that $|F| \leq p/2$. Then, for each node $v \in F$, during each consecutive $2|F|$ stages, there are at least $|F|$ steps, each in a different stage, during which v broadcasts as the only element of F.*

Proof. Let $k = |F|$. Let $T_1, \ldots, T_{2k}$ be the lines including v used during $2k$ consecutive stages. Then $T_1 - \{v_1\}, \ldots, T_{2k} - \{v_1\}$ are all disjoint. By the pigeonhole principle, at most $k - 1$ of them include elements from F.

Lemma 4. *Let F be the active nodes at the beginning of stage t. If $|F| \leq p/2$ then the average progress per stage during $2|F|$ stages starting from t is $\Omega(1)$.*

Proof. Let $k = |F|$. Consider the stages $T = \{t, t + 1, \ldots, t + 2k - 1\}$. If each of the nodes in F broadcasts during T as the only active node then the average progress per stage during T is at least $1/2$. Suppose this is not true and let $v \in F$ be such that whenever it broadcasts in T then some other active node broadcasts simultaneously. By Lemma 3, there are at least k steps, each in a different stage in T, such that v broadcasts during these steps as the only element of F. If some active v' broadcasts simultaneously then v' is not in F. Any two nodes broadcast together exactly once during all the stages. Hence if v is blocked from being the only active broadcasting node during k stages by nodes outside F, there need to be at least k distinct such nodes. None of them was active at the beginning of stage t, hence all of them acquired the active status during T. This again gives average progress of at least $1/2$ per stage during T.

Theorem 4. *Algorithm* C *completes broadcasting in time* $\mathcal{O}(n^{3/2})$.

Proof. We estimate the time needed to make progress $2n-1$. If at any stage the number a of active nodes is at most $p/2$ then we have a constant progress averaged over the next $2a$ stages by Lemma 4. If the number of active nodes is greater than $p/2$ at any stage then the round-robin will contribute at least $p/2 = \Omega(\sqrt{n})$ during the next $\mathcal{O}(n)$ steps. In either case the average progress per step is $\Omega(n^{-1/2})$. Hence after $\mathcal{O}(n/n^{-1/2}) = \mathcal{O}(n^{3/2})$ steps all the nodes are passive, and so have learned the source message.

6 Discussion

Our algorithms were presented as if the nodes knew the size of the network. This assumption can be avoided if the broadcasting is without acknowledgement and the protocol works in time $p(n)$, that is polynomial in the size n of the network, which is the case in this paper. The modified algorithm works in consecutive intervals of length $\mathcal{O}(p(2^i))$, with only the first 2^i nodes participating in the i-th interval. The total time in which broadcasting is completed is again $\mathcal{O}(p(n))$.

References

1. N. Alon, A. Bar-Noy, N. Linial and D. Peleg, A lower bound for radio broadcast, *Journal of Computer and System Sciences* 43 (1991) 290–298.
2. N. Alon, A. Bar-Noy, N. Linial and D. Peleg, Single round simulation of radio networks, *Journal of Algorithms* 13 (1992) 188–210.
3. R. Bar-Yehuda, O. Goldreich, and A. Itai, Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection, *Distributed Computing* 5 (1991) 67–72.
4. R. Bar-Yehuda, O. Goldreich, and A. Itai, On the time complexity of broadcast in radio networks: An exponential gap between determinism and randomization, *Journal of Computer and System Sciences* 45 (1992) 104–126.
5. R. Bar-Yehuda, A. Israeli, and A. Itai, Multiple communication in multi-hop radio networks, *SIAM Journal on Computing* 22 (1993) 875–887.
6. B.S. Chlebus, L. Gąsieniec, A.M. Gibbons, A. Pelc, and W. Rytter, Deterministic broadcasting in unknown radio networks, in *Proc. 11th Ann. ACM-SIAM Symp. on Discrete Algorithms*, San Francisco, California, 2000, pp. 861–870.
7. I. Chlamtac and S. Kutten, On broadcasting in radio networks - problem analysis and protocol design, *IEEE Transactions on Communications* 33 (1985) 1240–1246.
8. K. Diks, E. Kranakis, D. Krizanc and A. Pelc, The impact of knowledge on broadcasting time in radio networks, in *Proc. 7th European Symposium on Algorithms*, Prague, Czech Republic, 1999, Springer LNCS 1643, pp. 41–52.
9. I. Gaber and Y. Mansour, Broadcast in radio networks, in *Proc. 6th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1995, pp. 577–585.
10. R. Gallager, A perspective on multiaccess channels, *IEEE Trans. on Information Theory* 31 (1985) 124–142.
11. L.A. Goldberg, M. Jerrum, S. Kannan, and M. Paterson, A bound on the capacity of backoff and acknowledgement-based protocols, in this volume of Springer LNCS.

12. E. Kranakis, D. Krizanc and A. Pelc, Fault-tolerant broadcasting in radio networks, in *Proc. 6th European Symposium on Algorithms*, Venice, Italy, 1998, Springer LNCS 1461, pp. 283–294.
13. E. Kushilevitz and Y. Mansour, An $\Omega(D \lg(N/D))$ lower bound for broadcast in radio networks, *SIAM Journal on Computing* 27 (1998) 702–712.
14. E. Kushilevitz and Y. Mansour, Computation in noisy radio networks, in *Proc. 9th Ann. ACM-SIAM Symp. on Discrete Algorithms*, 1998, pp. 236–243.
15. K. Pahlavan and A. Levesque, "Wireless Information Networks," Wiley-Interscience, New York, 1995.
16. A. Sen and M. L. Huson, A new model for scheduling packet radio networks, in *Proc. 15th Ann. Joint Conference of the IEEE Computer and Communication Societies*, 1996, pp. 1116–1124.

An ω-Complete Equational Specification of Interleaving ⋆

Extended Abstract

W.J. Fokkink[1] and S.P. Luttik[1,2,⋆⋆]

[1] CWI, P.O. Box 94079, NL-1090 GB Amsterdam, The Netherlands
Wan.Fokkink@cwi.nl

[2] Department of Computer Science, University of Amsterdam
Kruislaan 403, NL-1098 SJ Amsterdam, The Netherlands
Bas.Luttik@cwi.nl

Abstract. We consider the process theory PA that includes an operation for parallel composition, based on the interleaving paradigm. We prove that the standard set of axioms of PA is not ω-complete by providing a set of axioms that are valid in PA, but not derivable from the standard ones. We prove that extending PA with this set yields an ω-complete specification, which is finite in a setting with finitely many actions.

1 Introduction

The interleaving paradigm consists of the assumption that two atomic actions cannot happen at the same time, so that concurrency reduces to nondeterminism. To express the concurrent execution of processes, many process theories have been accomodated with an operation for parallel composition that behaves according to the interleaving paradigm. For instance, CCS (see, e.g., Milner (1989)) has a binary operation for parallel composition —we shall denote it by $_ \,||\, _$— that satisfies the so-called *Expansion Law*:

$$\text{if } p = \sum_{i=1}^{m} a_i.p_i \text{ and } q = \sum_{j=1}^{n} b_j.q_j, \text{ then } p \,||\, q \approx \sum_{i=1}^{m} a_i.(p_i \,||\, q) + \sum_{j=1}^{n} b_j.(q_j \,||\, p);$$

here the $a_i._$ and the $b_j._$ are unary operations that prefix a process with an atomic action, and summation denotes a nondeterministic choice between its arguments.

The Expansion Law generates an infinite set of equations, one for each pair of processes p and q. Bergstra and Klop (1984) enhanced the equational characterisation of interleaving. They replaced action prefixing with a binary operation

⋆ A full version that contains the omitted proofs is available as CWI Technical Report; see http://www.cwi.nl/~luttik/

⋆⋆ Research supported by the Netherlands Organization for Scientific Research (NWO) under contract SION 612-33-008.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 729–743, 2000.
© Springer-Verlag Berlin Heidelberg 2000

$_ \cdot _$ for sequential composition and added an auxiliary operation $\lfloor\!\lfloor$ (the *left merge*; it is similar to $\|$, except that it must start execution with a step from its left argument). Their axiomatisation is finite for settings with finitely many atomic actions. Moller (1990) proved that interleaving is not finitely axiomatisable without an auxiliary operation such as the left merge.

The axioms of Bergstra and Klop (1984) form a ground-complete axiomatisation of bisimulation equivalence; ground terms p and q are provably equal if, and only if, they are bisimilar. Thus, it reflects for a large part our intuition about interleaving. On the other hand, it is not optimal. For instance, it can be shown by means of structural induction that every ground instance of the axiom $x \,\|\, (y \,\|\, z) \approx (x \,\|\, y) \,\|\, z$ is derivable (see Baeten and Weijland (1990)); however, the axiom itself is not derivable.

If an equational specification E has the property that $\mathsf{E} \vdash t^\sigma \approx u^\sigma$ for all ground substitutions σ implies that $\mathsf{E} \vdash t \approx u$, then E is called *ω-complete* (or: *inductively closed*). To derive any equation from such an equational specification it is never needed to use additional proof techniques such as structural induction. Therefore, in applications dealing with theorem proving, ω-completeness is a desirable property to have (see Lazrek *et al.* (1990)). In Heering (1986) it was argued that ω-completeness is desirable for the partial evaluation of programs.

Moller (1989) obtained an ω-complete axiomatisation for CCS without communication, by adding a law for *standard concurrency*:

$$(x \lfloor\!\lfloor y) \lfloor\!\lfloor z \approx x \lfloor\!\lfloor (y \,\|\, z).$$

In this paper we shall address the question whether PA, the subtheory of ACP without communication and encapsulation, is ω-complete. While the algebra studied by Moller (1989) has sequential composition in the form of prefix multiplication, PA incorporates the (more general) binary operation $\cdot$ for sequential composition. Having this operation, it is no longer sufficient to add the law for standard concurrency to arrive at an ω-complete axiomatisation. However, surprisingly, it is sufficient to add this law and the set of axioms generated by a single scheme:

$$(x \cdot \alpha \lfloor\!\lfloor \alpha) \approx (x \lfloor\!\lfloor \alpha) \cdot \alpha,$$

where α ranges over alternative compositions of distinct atomic actions; if the set of atomic actions is finite, then this scheme generates finitely many axioms.

An important part of our proof has been inspired by the excellent work of Hirshfeld and Jerrum (1999) on the decidability of bisimulation equivalence for normed process algebra. In particular, they distinguish two kinds of *mixed equations*, in which a parallel composition is equated to a sequential composition. The first kind consists of equations

$$(t \cdot \alpha^k) \,\|\, \alpha^l \approx (t \,\|\, \alpha^l) \cdot \alpha^k$$

for positive natural numbers k and l, and for sums of atomic actions α. These equations can be derived using standard concurrency and our new axioms. The

second kind of mixed equations are the so-called *pumpable equations*, which are of a more complex nature (see p. 419 of Hirshfeld and Jerrum (1999)). Basically, we show that there cannot exist pumpable equations that contain variables by associating with every candidate $t \approx u$ a ground substitution σ such that $t^\sigma \not\approx u^\sigma$.

The notion of ω-completeness is related to *action refinement*, where each atomic action may be refined to an arbitrary process. That is, in a theory with action refinement, the actions take over the role played by variables in our theory; the actions, as they occur in our theory, are not present in theories for action refinement. Aceto and Hennessy (1993) presented a complete axiomatisation for PA (including a special constant `nil`, being a hybrid of deadlock and empty process) with action refinement, modulo timed observational equivalence from Hennessy (1988). In this setting, laws such as $a \mathbin{\lfloor\!\lfloor} x \approx a \cdot x$, which hold in standard PA, are no longer valid, as the atomic action a can be refined into any other process.

This paper is set up as follows. In §2 we introduce the standard axioms of interleaving, and we prove that they do not form an ω-complete specification by proving that all ground substitution instances of our new axioms are derivable, while the axioms themselves are not. In §3 we state some basic facts about the theory of interleaving that we shall need in our proof of ω-completeness. In §4 we collect some results on certain mixed equations, and in §5 we investigate a particular kind of terms that consist of nestings of parallel and sequential compositions. In §6 we prove our main theorem, that the standard theory of interleaving enriched with the law for standard concurrency and our new axioms is ω-complete.

2 Interleaving

A *process algebra* is an algebra that satisfies the axioms A1–A5 of Table 1. Suppose that A is a set of constant symbols and suppose that $\|$ and $\mathbin{\lfloor\!\lfloor}$ are binary operation symbols; a process algebra with interpretations for the constant symbols in A and the operations $\|$ and $\mathbin{\lfloor\!\lfloor}$ satisfying M1, M4, M5, M2$_a$ and M3$_a$ for all $a \in A$, and M6$_\alpha$ for all sums of distinct elements of A, we shall call an *A-merge algebra*; the variety of A-merge algebras we denote by PA_A.

Table 1. The axioms of PA_A, with $a \in A$ and α any sum of distinct elements of A.

(A1)	$x + y$	$\approx y + x$	(M1)	$x \mathbin{\|} y$	$\approx x \mathbin{\lfloor\!\lfloor} y + y \mathbin{\lfloor\!\lfloor} x$
(A2)	$x + (y + z)$	$\approx (x + y) + z$	(M2$_a$)	$a \mathbin{\lfloor\!\lfloor} x$	$\approx a \cdot x$
(A3)	$x + x$	$\approx x$	(M3$_a$)	$a \cdot x \mathbin{\lfloor\!\lfloor} y$	$\approx a \cdot (x \mathbin{\|} y)$
(A4)	$(x + y) \cdot z$	$\approx x \cdot z + y \cdot z$	(M4)	$(x + y) \mathbin{\lfloor\!\lfloor} z$	$\approx x \mathbin{\lfloor\!\lfloor} z + y \mathbin{\lfloor\!\lfloor} z$
(A5)	$(x \cdot y) \cdot z$	$\approx x \cdot (y \cdot z)$	(M5)	$(x \mathbin{\lfloor\!\lfloor} y) \mathbin{\lfloor\!\lfloor} z$	$\approx x \mathbin{\lfloor\!\lfloor} (y \mathbin{\|} z)$
			(M6$_\alpha$)	$x \cdot \alpha \mathbin{\lfloor\!\lfloor} \alpha$	$\approx (x \mathbin{\lfloor\!\lfloor} \alpha) \cdot \alpha$

The axioms A1–A5 together with the axioms M1–M4 form the standard axiomatisation of interleaving. Consider the single-sorted signature Σ with the elements of A as constants and the binary operations $+$, $\cdot$, $\lfloor\!\lfloor$ and $||$. In writing terms we shall often omit the operation $\cdot$ for sequential composition; we assume that sequential composition binds strongest and that the operation $+$ for alternative composition binds weakest.

Let $\mathcal{R}$ consist of the axioms A3–A5 and M1–M4 of Table 1 interpreted as rewrite rules by orienting them from left to right. The term rewriting system $\langle \Sigma, \mathcal{R} \rangle$ is ground terminating and ground confluent modulo associativity and commutativity of $+$ (cf. the axioms A1 and A2). A ground term t is a *basic term* if there exist disjoint finite sets I and J, elements a_i and b_j of A and basic terms t_i, for $i \in I$ and $j \in J$ such that

$$t \approx \sum_{i \in I} a_i t_i + \sum_{j \in J} b_j \qquad \text{(by A1 and A2).}$$

Every ground normal form of $\langle \Sigma, \mathcal{R} \rangle$ is a basic term.

It is well-known that the axioms A1–A5 together with M1–M4 do not constitute an ω-complete axiomatisation; all ground substitution instances of M5 are derivable, while the axiom itself is not. Moller (1989) has shown that, in a setting with prefix sequential composition instead of the binary operation $\cdot$, it suffices to add M5 to obtain an ω-complete axiomatisation (see Groote (1990) for an alternative proof). Clearly, neither $x\alpha \lfloor\!\lfloor \alpha$ nor $(x \lfloor\!\lfloor \alpha)\alpha$ is an instance of any of the axioms A1–A5 and M1–M5, so $\mathrm{M6}_\alpha$ is not derivable. However, each ground substitution instance of $\mathrm{M6}_\alpha$ is derivable.

Proposition 1. *If α is a finite sum of elements of A, then, for every ground term t,*

$$\mathrm{A1}, \ldots, \mathrm{A5}, \mathrm{M1}, \ldots, \mathrm{M4} \vdash t\alpha \lfloor\!\lfloor \alpha \approx (t \lfloor\!\lfloor \alpha)\alpha.$$

Consequently, in the case of binary sequential composition, the axioms A1–A5 together with M1–M5 do not constitute an ω-complete axiomatision. In the sequel, we shall prove that PA_A is ω-complete.

3 Basic Facts

In every A-merge algebra, $+$ and $||$ are commutative and associative; we shall often implicitly make use of this. Also, we shall frequently abbreviate the statement $\mathsf{PA}_A \vdash u \approx u + t$ by $t \preccurlyeq u$; if $t \preccurlyeq u$, then we call t a *summand* of u. Note that $\preccurlyeq$ is a partial order on the set of terms modulo $\approx$; in particular, if $t \preccurlyeq u$ and $u \preccurlyeq t$, then $t \approx u$.

Lemma 2. *Let a be an element of A and let t, u and v be ground terms. If $at \preccurlyeq u + v$, then $at \preccurlyeq u$ or $at \preccurlyeq v$.*

Suppose t is a ground normal form of the system $\langle \Sigma, \mathcal{R} \rangle$ and suppose that

$$t \approx \sum_{i \in I} a_i t_i + \sum_{j \in J} b_j;$$

then the *degree* $d(t)$ of t is defined by $d(t) = |I| + |J|$. We let the degree of an arbitrary ground term be the degree of its unique normal form in $\langle \Sigma, \mathcal{R} \rangle$. By $d_{\max}(t)$ we shall denote the maximal degree that occurs in t, i.e.,

$$d_{\max}(t) = \max(\{d(t)\} \cup \{d_{\max}(t') \mid \text{there exists an } a \in A \text{ such that } at' \preccurlyeq t\}).$$

Lemma 3. *If α is a finite sum of elements of A, then*

$$\mathsf{PA}_A \vdash x\alpha \mathbin{\lfloor\!\lfloor} \alpha^n \approx (x \mathbin{\lfloor\!\lfloor} \alpha^n)\alpha, \text{ and } \mathsf{PA}_A \vdash x\alpha \mathbin{||} \alpha^n \approx (x \mathbin{||} \alpha^n)\alpha.$$

Proof. It is straightforward to show by induction on n that the identity (*) $\alpha^{n+1} \approx \alpha^n \mathbin{||} \alpha$ is derivable from PA_A; we shall use it in the proof of the first set of equations (**) $x\alpha \mathbin{\lfloor\!\lfloor} \alpha^n \approx (x \mathbin{\lfloor\!\lfloor} \alpha^n)\alpha$, which is by induction on n. If $n = 1$, then (**) is an instance of $\mathrm{M6}_\alpha$, and for the induction step we have the following derivation:

$$\begin{aligned}
x\alpha \mathbin{\lfloor\!\lfloor} \alpha^{n+1} &\approx x\alpha \mathbin{\lfloor\!\lfloor} (\alpha^n \mathbin{||} \alpha) && \text{(by *)}\\
&\approx (x\alpha \mathbin{\lfloor\!\lfloor} \alpha^n) \mathbin{\lfloor\!\lfloor} \alpha && \text{(by M5)}\\
&\approx (x \mathbin{\lfloor\!\lfloor} \alpha^n)\alpha \mathbin{\lfloor\!\lfloor} \alpha && \text{(by IH)}\\
&\approx ((x \mathbin{\lfloor\!\lfloor} \alpha^n) \mathbin{\lfloor\!\lfloor} \alpha)\alpha && \text{(by M6}_\alpha\text{)}\\
&\approx (x \mathbin{\lfloor\!\lfloor} (\alpha^n \mathbin{||} \alpha))\alpha && \text{(by M5)}\\
&\approx (x \mathbin{\lfloor\!\lfloor} \alpha^{n+1})\alpha && \text{(by *).}
\end{aligned}$$

The second set of equations is also derived by induction on n, using (**). □

Milner and Moller (1993) proved that if t, u and v are ground terms such that $t \mathbin{||} v$ and $u \mathbin{||} v$ are bisimilar, then t and u are bisimilar (a similar result was obtained earlier by Castellani and Hennessy (1989) in the context of distributed bisimulation). Also, they proved that every finite process has, up to bisimulation equivalence, a unique decomposition into prime components. Since PA_A is a sound and complete axiomatisation for bisimulation equivalence (Bergstra and Klop, 1984), the following two results are consequences of theirs.

Lemma 4. *If t, u and v are ground terms such that $\mathsf{PA}_A \vdash t \mathbin{||} v \approx u \mathbin{||} v$, then $\mathsf{PA}_A \vdash t \approx u$.*

Definition 5. *A ground term t we shall call* parallel prime *if there do not exist ground terms u and v such that $\mathsf{PA}_A \vdash t \approx u \mathbin{||} v$.*

Theorem 6 (Unique factorisation). *Any ground term can be expressed uniquely as a parallel composition of parallel prime components.*

We associate to each term t a *norm* $\lfloor t \rfloor$ and a *depth* $\lceil t \rceil$ as follows:

$$\begin{array}{lll} \lfloor x \rfloor = \lfloor a \rfloor = 1 & \lceil a \rceil = \lceil x \rceil = 1 & (a \in A \text{ and } x \text{ a variable}); \\ \lfloor x * y \rfloor = \lfloor x \rfloor + \lfloor y \rfloor & \lceil x * y \rceil = \lceil x \rceil + \lceil y \rceil & \text{if } * \in \{\cdot, \lfloor\!\lfloor, ||\}; \text{ and} \\ \lfloor x + y \rfloor = \min\{\lfloor x \rfloor, \lfloor y \rfloor\} & \lceil x + y \rceil = \max\{\lceil x \rceil, \lceil y \rceil\}. & \end{array}$$

Notice that if $t \approx u$, then t and u must have equal norm and depth.

Lemma 7. *If t, t', u and u' are ground terms such that $\lfloor t \rfloor = \lfloor t' \rfloor$, $\lfloor u \rfloor = \lfloor u' \rfloor$ and $\mathsf{PA}_A \vdash tu \approx t'u'$, then $\mathsf{PA}_A \vdash t \approx t'$ and $\mathsf{PA}_A \vdash u \approx u'$.*

Definition 8. *Let t and t' be ground terms; we shall write $t \longrightarrow t'$ if there exists $a \in A$ such that $at' \preccurlyeq t$ and $\lfloor t' \rfloor < \lfloor t \rfloor$. We define the set $\mathsf{red}(t)$ of* reducts *of t as the least set that contains t and is closed under $\longrightarrow$; if $t \longrightarrow t'$, then we call t' an* immediate reduct *of t.*

Lemma 9. *Let t be a ground term. If $t \longrightarrow t'$ and $t \longrightarrow t''$ implies that $\mathsf{PA}_A \vdash t' \approx t''$ for all ground terms t' and t'', then there exists a parallel prime ground term t^* such that $\mathsf{PA}_A \vdash t \approx t^* \,||\, \ldots \,||\, t^*$.*

Proof. First, suppose that u and v are parallel prime, and let u' and v' be such that $u \longrightarrow u'$ and $v \longrightarrow v'$; then, $u \,||\, v \longrightarrow u' \,||\, v$ and $u \,||\, v \longrightarrow u \,||\, v'$. So, if $u' \,||\, v \approx u \,||\, v'$, then since $\lfloor u' \rfloor < \lfloor u \rfloor$, u cannot be a component of the prime decomposition of u'; hence, by Theorem 6, $u \approx v$.

Suppose $t \approx t_1 \,||\, \ldots \,||\, t_n$, with t_i parallel prime for all $1 \leq i \leq n$ and $\lfloor t_1 \rfloor \leq \cdots \leq \lfloor t_n \rfloor$.

If $\lfloor t_1 \rfloor = 1$, then $\lfloor t_i \rfloor = 1$ for all $1 \leq i \leq n$; for suppose that t_i' is a ground term such that $t_i \longrightarrow t_i'$, then from $t_2 || \cdots || t_n \approx t_1 || \cdots || t_{i-1} || t_i' || t_{i+1} || \cdots || t_n$, we get by Lemma 4 that $t_i \approx t_1 || t_i'$, but t_i is parallel prime. From $t_1 || \cdots || t_{i-1} || t_{i+1} || \cdots || t_n \approx t_1 \,||\, \cdots \,||\, t_{j-1} \,||\, t_{j+1} \,||\, \cdots \,||\, t_n$, we conclude by Lemma 4 that $t_i \approx t_j$.

The remaining case is that $\lfloor t_i \rfloor > 1$ for all $1 \leq i \leq n$. Let t_i' and t_j' be ground terms such that $t_i \longrightarrow t_i'$ and $t_j \longrightarrow t_j'$ for some $1 \leq i < j \leq n$; then by Lemma 4 $t_i' \,||\, t_j \approx t_i \,||\, t_j'$. Since $\lfloor t_i' \rfloor < \lfloor t_i \rfloor$, t_i cannot be a component of the prime decomposition of t_i', so by Theorem 6 $t_i \approx t_j$. □

4 Mixed Equations

We shall collect some results about *mixed equations*; these are equations of the form $tu \approx v \,||\, w$.

Lemma 10. *If t, u and v are ground terms such that $\mathsf{PA}_A \vdash tu \approx u \,||\, v$, then there exists a finite sum α of elements of A such that $\mathsf{PA}_A \vdash u \approx \alpha^k$ for some $k \geq 1$.*

Proof. Note that $\lceil t \rceil = \lceil v \rceil$; we shall first prove the following
Claim: if $\lceil t \rceil, \lceil v \rceil = 1$, then there exists a $k \geq 1$ such that $u \approx t^k$ and $t \approx v$.

Let $t = a_1 + \ldots + a_m$ with $a_1, \ldots, a_m \in A$; we proceed by induction on $\lfloor u \rfloor$.

If $\lfloor u \rfloor = 1$, then there exists $a \in A$ such that $a \preccurlyeq u$, whence $at \preccurlyeq u \,||\, v$. Since $a_1 u + \cdots + a_m u \approx u \,||\, v$, there exists by Lemma 2 an i such that $a_i u \approx av$; hence by Lemma 7 $u \approx v$. Since $\lceil v \rceil = 1$ it follows that $tu \approx v \,||\, v \approx vv \approx vu$, hence by Lemma 7 $t \approx v$.

If $\lfloor u \rfloor > 1$, then there exist $b_1, \ldots, b_n \in A$ and ground terms $u_1, \ldots, u_n$ such that $u \approx b_1 u_1 + \ldots + b_n u_n$. Then $a_1 u + \ldots + a_m u \approx tu \approx u \,||\, v \approx b_1(u_1 \,||\, v) + \ldots + b_n(u_n \,||\, v) + vu$, so by Lemma 2 $u_i \,||\, v \approx u$, for all $1 \leq i \leq n$. By Lemma 4 there exists u' such $u_i \approx u'$ for all $1 \leq i \leq n$, and, by A4, $(b_1 + \ldots + b_n)u' \approx u \approx u' \,||\, v$. Hence by the induction hypothesis $v \approx b_1 + \ldots + b_n$ and $u' \approx v^k$ for some $k \geq 1$. So $u \approx v^{k+1}$, and from

$$\begin{aligned} tu &\approx vu + b_1(u' \,||\, v) + \cdots + b_n(u' \,||\, v) \\ &\approx vu + vu && \text{(by A4)} \\ &\approx vu && \text{(by A3)} \end{aligned}$$

it follows, by Lemma 7, that $t \approx v$. This completes the proof of our claim.

The proof of the lemma is by induction on $\lceil v \rceil$. If $\lceil t \rceil, \lceil v \rceil = 1$ then t is a finite sum of elements of A and by our claim $u \approx t^k$ for some $k \geq 1$. If $\lceil t \rceil, \lceil v \rceil > 1$, then there exists $a \in A$ and ground terms t' and v' such that $av' \preccurlyeq v$ and $t'u \approx u \,||\, v'$; hence, by the induction hypothesis, there exists a finite sum α of elements of A such that $u \approx \alpha^k$, for some $k \geq 1$. □

Lemma 10 has the following consequence.

Lemma 11. *If t, t', u and v are ground terms such that $\mathsf{PA}_A \vdash tu \approx t'u \,||\, v$, then there exists a finite sum α of elements of A such that $\mathsf{PA}_A \vdash u \approx \alpha^k$ for some $k \geq 1$.*

Lemma 12. *Let α be a finite sum of elements of A; if t, u and v are ground terms such that $\mathsf{PA}_A \vdash t\alpha^k \approx u \,||\, v$ for some $k \geq 1$, then $\mathsf{PA}_A \vdash u \approx \alpha^l$ for some $l \leq k$, or there exists a ground term t' such that $\mathsf{PA}_A \vdash u \approx t'\alpha^k$.*

Proof. The proof is by induction on the norm of v.

If $\lfloor v \rfloor = 1$, then there exists an $a \in A$ such that $a \preccurlyeq v$, whence $au \preccurlyeq t\alpha^k$. If $a \preccurlyeq t$, then $u \approx \alpha^k$, and if there exists a ground term t' such that $at' \preccurlyeq t$, then $u \approx t'\alpha^k$.

Suppose that $\lfloor v \rfloor > 1$ and let v' be a ground term such that $\lfloor v' \rfloor < \lfloor v \rfloor$ and $av' \preccurlyeq v$, whence $a(u \,||\, v') \preccurlyeq t\alpha^k$. If $a \preccurlyeq t$, then $u \,||\, v' \approx \alpha^k$, hence there exists an $l < k$ such that $u \approx \alpha^l$. Otherwise, suppose that t^* is a ground term such that $at^* \preccurlyeq t$ and $u \,||\, v' \approx t^*\alpha^k$; by induction hypothesis $u \approx \alpha^l$ for some $l \leq k$, or there exists a ground term t' such that $u \approx t'\alpha^k$. □

Hirshfeld and Jerrum (1998) give a thorough investigation of a particular kind of mixed equations; we shall adapt some of their theory to our setting.

Let α be a finite sum of elements of A. A ground term t we shall call *α-free* if $t \not\approx \alpha$ and there exists no ground term t' such that $t \approx t' \,||\, \alpha$. We shall call a ground term t an *α-term* if $t \approx \alpha^k$ for some $k \geq 1$. The *α-norm* $\lfloor t \rfloor_\alpha$ of a ground term t is the length of the shortest reduction of t to an α-term, or the norm of t if such a reduction does not exist. Note that if $t \approx u$, then $\lfloor t \rfloor_\alpha = \lfloor u \rfloor_\alpha$; the α-norm of an equation is the α-norm of both sides. We shall write $t \longrightarrow_\alpha t'$ if $t \longrightarrow t'$ and $\lfloor t' \rfloor_\alpha < \lfloor t \rfloor_\alpha$; if $\lfloor u \rfloor_\alpha = 1$, then we say that u is an *α-unit*. In line with Definition 8, a ground term t' is an *α-reduct* of a ground term t if t' is reachable from t by an α-reduction.

It is easy to see that $\lfloor t \,||\, u \rfloor_\alpha = \lfloor t \rfloor_\alpha + \lfloor u \rfloor_\alpha$, so we have the following lemma.

Lemma 13. *Let α be a finite sum of elements of A; any α-free α-unit is parallel prime.*

Lemma 14. *If t is α-free, then $t\alpha$ is α-free.*

Hirshfeld and Jerrum (1998) proved a variant of Lemma 9.

Lemma 15. *Let α be a finite sum of elements of A, and let t be an α-free ground term. If $t \longrightarrow_\alpha t'$ and $t \longrightarrow_\alpha t''$ implies that $\mathsf{PA}_A \vdash t' \approx t''$ for all ground terms t' and t'', then there exists a parallel prime ground term t^* such that $\mathsf{PA}_A \vdash t \approx t^* \,||\, \ldots \,||\, t^*$.*

A *pumpable equation* is a mixed equation of the form

$$(t_1 \,||\, \cdots \,||\, t_m)\alpha^k \approx u_1\alpha^k \,||\, \cdots \,||\, u_n\alpha^k,$$

where α is a finite sum of elements of A, $k \geq 1$, $m, n \geq 2$ and t_i and u_j are α-free ground terms for $1 \leq i \leq m$ and $1 \leq j \leq n$. The following lemma occurs in Hirshfeld and Jerrum (1998) as Lemma 7.2.

Lemma 16. *There are no pumpable equations with α-norm less than three.*

Proposition 17. *Let t, u, u' and v be ground terms such that t and v are α-free and*

$$\mathsf{PA}_A \vdash (t \,||\, u)\alpha^k \approx v\alpha^k \,||\, u'\alpha^k. \tag{1}$$

If u and u' are α-units, then $\mathsf{PA}_A \vdash u \approx u'$.

Proof. If there exists a ground term u^* such that $u \approx u^* \,||\, \alpha$, then by Lemma 3 $v\alpha^k \,||\, u'\alpha^k \approx (t \,||\, u^* \,||\, \alpha)\alpha^k \approx (t \,||\, u^*)\alpha^k \,||\, \alpha$; by Lemma 14 $v\alpha^k$ is α-free, hence there exists a ground term u^{**} such that $u' \approx u^{**} \,||\, \alpha$. Vice versa, from $u' \approx u^{**} \,||\, \alpha$ we obtain the existence of a u^* such that $u \approx u^* \,||\, \alpha$. In both cases $(t \,||\, u^*)\alpha^k \,||\, \alpha \approx v\alpha^k \,||\, u^{**}\alpha^k \,||\, \alpha$, whence

$$(t \,||\, u^*)\alpha^k \approx v\alpha^k \,||\, u^{**}\alpha^k.$$

Hence, we may assume without loss of generality that the α-units u and u' are α-free, so that (1) is a *pumpable equation*. By Lemma 16 there are no pumpable equations with α-norm less than three, so $\lfloor t \rfloor_\alpha, \lfloor v \rfloor_\alpha \geq 2$; we prove the lemma by induction on $\lfloor t \rfloor_\alpha$.

If there exist ground terms t' and v' such that $t \longrightarrow_\alpha t'$, $v \longrightarrow_\alpha v'$ and $(t' \,||\, u)\alpha^k \approx v'\alpha^k \,||\, u'\alpha^k$, then we may conclude $u \approx u'$ from the induction hypothesis. Since the α-units $u'\alpha^k$ and u have unique immediate α-reducts, in the case that remains, t and v have unique immediate α-reducts t' and v', respectively; hence, by Lemma 15 there exists a parallel prime ground term v^* such that $v \approx v^* \,||\, \cdots \,||\, v^*$. By Lemma 3

$$(t' \,||\, u)\alpha^k \approx v\alpha^k \,||\, \alpha^{k+i} \approx (v \,||\, \alpha^{k+i})\alpha^k, \text{ for some } i \geq 0,$$

so $t' \,||\, u \approx v^* \,||\, \cdots \,||\, v^* \,||\, \alpha^{k+i}$. Since u is α-free, whence parallel prime by Lemma 14, it follows that $u \approx v^*$; hence

$$(t \,||\, u)\alpha^k \approx (u \,||\, \cdots \,||\, u)\alpha^k \,||\, u'\alpha^k \text{ and } t' \approx u \,||\, \cdots \,||\, u \,||\, \alpha^{k+i}.$$

Clearly, there exists a $j \geq k$ such that $u'\alpha^k \,||\, \alpha^j$ is an α-reduct of $v\alpha^k \,||\, u'\alpha^k$ (α-reduce $v\alpha^k$ to α^j). Hence, by Lemma 3, $(u' \,||\, \alpha^j)\alpha^k$ is an α-reduct of $(t \,||\, u)\alpha^k$, so $u' \,||\, \alpha^j$ is an α-reduct of $t \,||\, u$. If $u' \,||\, \alpha^j$ is obtained by reducing t to an α-term, then $u \approx u'$ follows, since u and u' are α-free. Otherwise, there exists $j' \leq j$ such that $u' \,||\, \alpha^{j'}$ is an α-reduct of t, hence of the unique immediate α-reduct t' of t. Every α-reduct of t' with α-norm 1 is of the form $u \,||\, \alpha^{j''}$. Since u and u' are parallel prime, $u \approx u'$ follows. □

5 Mixed Terms

We shall now define the set of *head normal forms*, thus restricting the set of terms that we need to consider in our proof that PA_A is ω-complete. The syntactic form of head normal form motivate our investigation of a particular kind of terms that we shall call *mixed terms* (nestings of parallel and sequential compositions). We shall work towards a theorem that certain instantiations of mixed terms are either parallel prime or a parallel composition of a parallel prime and α^k for some finite sum α of elements of A.

Let x be a variable, suppose $t = x$ or $t = xt'$ for some term t', and suppose $\bar{u} = u_1, \ldots, u_j$ and $\bar{v} = v_1, \ldots, v_j$ are sequences of terms; we define the set of *x-prefixes* $L_j[t, \bar{u}, \bar{v}]$ inductively as follows:

$$L_0[t] = t; \text{ and}$$
$$L_{j+1}[t, \bar{u}, u_{j+1}, \bar{v}, v_{j+1}] = (L_j[t, \bar{u}, \bar{v}] \mathbin{\lfloor\!\lfloor} u_{j+1})v_{j+1}.$$

A term t is a *head normal form* if there exist finite sets I, J, K and L such that

$$t \approx \sum_{i \in I} a_i t_i + \sum_{j \in J} b_j + \sum_{k \in K} v_k \mathbin{\lfloor\!\lfloor} u_k + \sum_{l \in L} w_l \qquad \text{(by A1 and A2)}$$

with the a_i and b_j elements of A, the t_i and u_k arbitrary terms and each v_k and w_l an x-prefix for some variable x.

Lemma 18. *For each term t there exists a head normal form t^* such that* $\mathsf{PA}_A \vdash t \approx t^*$.

We shall associate with every equation $t \approx u$ a substitution σ such that $t^\sigma \approx u^\sigma$ implies that $t \approx u$. The main idea of our ω-completenss proof is to substitute for every variable in t or u a ground term that has a subterm φ_n of degree n, where, intuitively, n is large compared to the degrees already occurring in t and u. Let a be an element of A and let $n \geq 1$; we define

$$\varphi_n = a^n + a^{n-1} + \ldots + a.$$

Lemma 19. *If $n \geq 2$ and t is a ground term, then $\varphi_n t$ is parallel prime.*

Suppose t is a term, and let $\bar{u} = u_1, \ldots, u_j$ and $\bar{v} = v_1, \ldots, v_j$ be sequences of terms; we define the set of *mixed terms* $M_j[t, \bar{u}, \bar{v}]$ inductively as follows:

$$M_0[t] = t; \text{ and}$$
$$M_{j+1}[t, \bar{u}, u_{j+1}, \bar{v}, v_{j+1}] = (M_j[t, \bar{u}, \bar{v}] \mid\mid u_{j+1})v_{j+1}.$$

Let t be a ground term; we denote by $d^{\rightarrow}_{\max}(t)$ the least upperbound for the degrees of all the reducts of t, i.e.,

$$d^{\rightarrow}_{\max}(t) = \max\{d(t') \mid t' \in \mathsf{red}(t)\}.$$

Definition 20. *A mixed term $M_j[\varphi_n t, \bar{u}, \bar{v}]$ we shall call a* generalised φ_n-term *if*

$$d^{\rightarrow}_{max}(M_j[t, \bar{u}, \bar{v}]) < n.$$

Note that there are no generalised φ_1-terms.

Lemma 21. *Let $M_j[\varphi_n t, \bar{u}, \bar{v}]$ be a generalised φ_n-term and let u be a ground term such that*

$$\mathsf{PA}_A \vdash M_j[\varphi_n t, \bar{u}, \bar{v}] \approx \varphi_n t v_1 \cdots v_j \mid\mid u.$$

Then there exists a finite sum α of elements of A such that $\mathsf{PA}_A \vdash v_1 \cdots v_j \approx \alpha^k$ and $\mathsf{PA}_A \vdash u \approx u_1 \mid\mid \cdots \mid\mid u_j \approx \alpha^l$ for some $k, l \geq 1$.

Proof. From $M_j[\varphi_n t, \bar{u}, \bar{v}] \longrightarrow M_j[t, \bar{u}, \bar{v}]$ and $d(M_j[t, \bar{u}, \bar{v}]) < n$ it follows that $M_j[t, \bar{u}, \bar{v}] \approx t v_1 \cdots v_j \mid\mid u$; hence

$$d^{\rightarrow}_{\max}(t v_1 \cdots v_j \mid\mid u) < n. \tag{2}$$

Note that $\lfloor u \rfloor = \lfloor u_1 \,||\, \cdots \,||\, u_j \rfloor$; we shall prove the lemma by induction on $\lfloor u \rfloor$.

If $\lfloor u \rfloor = 1$, then $j = 1$ and $\lfloor u_1 \rfloor = 1$. By Lemma 11 there exists a finite sum α of elements of A such that $v_1 \approx \alpha^k$ for some $k \geq 1$, and hence by Lemma 12 $u \approx \alpha$. By Lemma 3 $(\varphi_n t \,||\, u_1)\alpha^k \approx \varphi_n t \alpha^k \,||\, \alpha \approx (\varphi_n t \,||\, \alpha)\alpha^k$, so $u_1 \approx \alpha$.

If $\lfloor u \rfloor > 1$, then there are three cases: $\lfloor u_j \rfloor = 1$ and $j > 1$, $\lfloor u_j \rfloor > 1$ and $j = 1$, and $\lfloor u_j \rfloor > 1$ and $j > 1$. We shall only treat the last case; for the other two cases the proof is similar.

Let u'_j be an immediate reduct of u_j; by (2) there exists an immediate reduct u' of u such that

$$M_j[\varphi_n t, u_1, \ldots, u_{j-1}, u'_j, \bar{v}] \approx \varphi_n t v_1 \cdots v_j \,||\, u'.$$

Since $M_j[\varphi_n t, u_1, \ldots, u_{j-1}, u'_j, \bar{v}]$, there exists by the induction hypothesis a finite sum α of elements of A such that $v_1 \cdots v_j \approx \alpha^k$ and $u_1 \,||\, \cdots \,||\, u_{j-1} \,||\, u'_j \approx \alpha^l$ for some $k, l \geq 1$. By means of $j-1$ applications of Lemma 3, we find that

$$M_j[\varphi_n t, \bar{u}, \bar{v}] \approx (\varphi_n t \alpha^{k - \lfloor v_j \rfloor} \,||\, u_j)\alpha^{\lfloor v_j \rfloor} \,||\, \alpha^l \approx \varphi_n t \alpha^k \,||\, u.$$

Since $\varphi_n t \alpha^k$ is parallel prime by Lemma 19, there exists a ground term u^* with $\lfloor u^* \rfloor = \lfloor u_j \rfloor$ and $\lfloor u^* \rfloor_\alpha = \lfloor u_j \rfloor_\alpha$ such that $u \approx u^* \,||\, \alpha^l$ and $(\varphi_n t \alpha^{k - \lfloor v_j \rfloor} \,||\, u_j)\alpha^{\lfloor v_j \rfloor} \approx \varphi_n t \alpha^k \,||\, u^*$. If $\lfloor u^* \rfloor \leq \lfloor v_j \rfloor$, then by Lemma 12 u^* is an α-term, which implies that u and u_j are also α-terms. So suppose that $\lfloor u^* \rfloor > \lfloor v_j \rfloor$, and let $u^\dagger$ be a ground term such that $u^* \approx u^\dagger \alpha^{\lfloor v_j \rfloor}$. Since $\lfloor u_j \rfloor > \lfloor u^\dagger \rfloor$ and $\lfloor u_j \rfloor_\alpha = \lfloor u^\dagger \rfloor_\alpha$, by Proposition 17, u_j and $u^\dagger$ are not α-units. Since u'_j is an α-term and $u_j \longrightarrow u'_j$, u_j must be an α-term, so also $u^\dagger$ is an α-term. Consequently, $u \approx u^* \,||\, \alpha^l \approx u^\dagger \alpha^{\lfloor v_j \rfloor} \,||\, \alpha^l$ is an α-term. $\square$

Lemma 22. *Let $M_j[\varphi_n t, \bar{u}, \bar{v}]$ be a generalised φ_n-term. If $M_j[\varphi_n t, \bar{u}, \bar{v}]$ is not α-free, then there exists $i \leq j$ such that $\mathsf{PA}_A \vdash v_i \cdots v_j \approx \alpha^k$ and u_i is not α-free.*

Proof. Let t^* be a ground term such that

$$M_j[\varphi_n t, \bar{u}, \bar{v}] \approx t^* \,||\, \alpha. \tag{3}$$

Since $\varphi_n t v_1 \cdots v_j$ is a reduct of $M_j[\varphi_n t, \bar{u}, \bar{v}]$ and by Lemma 19 $\varphi_n t v_1 \cdots v_j$ is parallel prime, $\varphi_n t v_1 \cdots v_j$ must be a reduct of t^*. Then $\varphi_n t v_1 \cdots v_j \,||\, \alpha$ is a reduct of $M_j[\varphi_n t, \bar{u}, \bar{v}]$, so there exist sequences of ground terms $\bar{u}'$ and $\bar{v}'$ such that for some $1 \leq j' \leq j$ and $1 \leq i \leq j$,

$$M_{j'}[\varphi_n t v_1 \cdots v_{i-1}, \bar{u}', \bar{v}'] \approx \varphi_n t v_1 \cdots v_j \,||\, \alpha, \text{ where } v'_1 \cdots v'_{j'} \approx v_i \cdots v_j.$$

By Lemma 21 $v_i \cdots v_j \approx \alpha^k$, so in particular $v_j \approx \alpha^l$ for some $l \leq k$. Clearly, $\lfloor t^* \rfloor > l$, so by Lemma 12 there exists $t^\dagger$ such that $t^* \approx t^\dagger \alpha^l$. We apply Lemma 3 to the right-hand side of (3) and cancel the α^l-tail on both sides to obtain

$$M_{j-1}[\varphi_n t, u_1, \ldots, u_{j-1}, v_1, \ldots, v_{j-1}] \,||\, u_j \approx t^\dagger \,||\, \alpha.$$

The remainder of the proof is by induction on j. If $j = 1$, then $\varphi_n t || u_j \approx t^\dagger || \alpha$ implies that u_j is not α-free and we are done. If $j > 1$ and u_j is α-free, then $M_{j-1}[\varphi_n t, u_1, \ldots, u_{j-1}, v_1, \ldots, v_{j-1}]$ is not α-free, so by the induction hypothesis there exists some $1 \leq i' \leq j-1$ such that $u_{i'}$ is not α-free and $v_{i'} \cdots v_{j-1} \approx \alpha^{k'}$, whence $v_{i'} \cdots v_j \approx \alpha^{k'+l}$. □

Proposition 23. *If a generalised φ_n-term t^* is not parallel prime, then there exists a finite sum α of elements of A and a parallel prime generalised φ_n-term $t^\dagger$ such that $t^* \approx t^\dagger \,||\, \alpha^k$ for some $k \geq 1$.*

Proof. Let $t^* \approx M_j[\varphi_n t, \bar{u}, \bar{v}]$ and let $t_1, \ldots, t_o$ be parallel prime ground terms such that

$$M_j[\varphi_n t, \bar{u}, \bar{v}] \approx t_1 \,||\, \ldots \,||\, t_o.$$

Since $\varphi_n t v_1 \cdots v_j$ is a reduct of $M_j[\varphi_n t, \bar{u}, \bar{v}]$ and parallel prime, $\varphi_n t v_1 \cdots v_j$ must be a reduct of some t_i $(1 \leq i \leq o)$; assume without loss of generality that it is a reduct of t_1.

Suppose that $M_j[\varphi_n t, \bar{u}, \bar{v}]$ is not parallel prime and let $u \approx t_2 \,||\, \cdots \,||\, t_o$. Since $\varphi_n t v_1 \cdots v_j \,||\, u$ is a reduct of $M_j[\varphi_n t, \bar{u}, \bar{v}]$, there exist sequences of ground terms $\bar{u}'$ and $\bar{v}'$ such that, for some $1 \leq j' \leq j$ and $1 \leq i \leq j$,

$$M_{j'}[\varphi_n t v_1 \cdots v_{i-1}, \bar{u}', \bar{v}'] \approx \varphi_n t v_1 \cdots v_j \,||\, u, \text{ where } v'_1 \cdots v'_{j'} \approx v_i \cdots v_j.$$

So by Lemma 21, there exists a finite sum α of elements of A such that $u \approx \alpha^k$, for some $k \geq 1$.

It remains to prove that if $M_j[\varphi_n t, \bar{u}, \bar{v}] \approx t^\dagger \,||\, \alpha$, then $t^\dagger$ is a generalised φ_n-term, for then it follows that t_1 is a generalised φ_n-term by induction on k. Since $M_j[\varphi_n t, \bar{u}, \bar{v}]$ is not α-free, there exists by Lemma 22 an $i \leq j$ such that u_i is not α-free and $v_i \cdots v_j \approx \alpha^l$ for some $l \geq 1$. So either $u_i \approx \alpha$ or there exists u'_i such that $u_i \approx u'_i \,||\, \alpha$; we only consider the second possibility, as the other can be dealt with similarly. By Lemma 3 we obtain

$$M_j[\varphi_n t, \bar{u}, \bar{v}] \approx M_{j-1}[\varphi_n t, u_1, \ldots, u_{i-1}, u'_i, u_{i+1}, \ldots, u_j, \bar{v}] \,||\, \alpha$$

Hence $t^\dagger \approx M_{j-1}[\varphi_n t, u_1, \ldots, u_{i-1}, u'_i, u_{i+1}, \ldots, u_j, \bar{v}]$ is a generalised φ_n-term. □

6 ω-Completeness

Let A be a nonempty set; we shall now prove that PA_A is ω-complete. We shall assume that the variables used in an equation $t \approx u$ are enumerated by $x_1, x_2, \ldots, x_k, \ldots$ Let x_i be a variable and let m be a natural number; the particular kind of substitutions σ_m that we shall use in our proof satisfy

$$\sigma_m(x_i) = a(a\varphi_{i+m} + a)a.$$

We want to choose m large compared to the degrees already occurring in t and u; with every term t we associate a natural number $d^{\sigma}_{\max}(t)$ that denotes the maximal degree that occurs in t after applying a substitution of the form described above, treating the terms φ_{i+m} as fresh constants.

Definition 24. *Suppose $\Xi = \{\xi_1, \xi_2, \ldots, \xi_k, \ldots\}$ is a countably infinite set of constant symbols such that $\Xi \cap A = \emptyset$. Let t be a term and let σ be a substitution such that*

$$\sigma(x_i) = a(a\xi_i + a)a.$$

We define $d^{\sigma}_{max}(t)$ as the maximal degree that occurs in t^{σ}, i.e., $d^{\sigma}_{max}(t) = d_{max}(t^{\sigma})$.

Lemma 25. *If t is a term and let m be a natural number, then*

i. $d^{\rightarrow}_{max}(t^{\sigma_m}) \leq d^{\sigma}_{max}(t)$*; and*
ii. if $a \in A$ and t' is a ground term such that $at' \preccurlyeq t^{\sigma_m}$, then $d(t') \leq d^{\sigma}_{max}(t)$.

If $L_j[t, \bar{u}, \bar{v}]$ is an x_i-prefix and m is a natural number, then

$$L_j[t, \bar{u}, \bar{v}]^{\sigma_m} \longrightarrow M_j[(a\varphi_{i+m} + a)t'', \bar{u}, \bar{v}]^{\sigma_m} \succcurlyeq aM_j[\varphi_{i+m}t'', \bar{u}, \bar{v}]^{\sigma_m}, \qquad (4)$$

where $t'' = a$ if $t = x_i$ and $t'' = at'$ if $t = x_i t'$ for some term t'. If $m \geq d^{\sigma}_{\max}(L_j[t, \bar{u}, \bar{v}])$, then $M_j[\varphi_{i+m}t'', \bar{u}, \bar{v}]^{\sigma_m}$ is a generalised φ_n-term; we shall call it the *generalised φ_{i+m}-term associated with $L_j[t, \bar{u}, \bar{v}]$ by σ_m.*

For ground terms t and t^*, let us write $t \mapsto t^*$ if there exists a ground term t' and an $a \in A$ such that $t \longrightarrow t' \succcurlyeq at^*$.

Lemma 26. *Let t be an x-prefix and suppose that $m \geq d^{\sigma}_{max}(t)$. If $n > m$ and t^* and $t^{\dagger}$ are generalised φ_n-terms such that $t^{\sigma_m} \mapsto t^*$ and $t^{\sigma_m} \mapsto t^{\dagger}$, then $\mathsf{PA}_A \vdash t^* \approx t^{\dagger}$.*

Proof. Note that the unique immediate reduct of t^{σ_m} is of the form $M_j[(a\varphi_{i+m} + a)t', \bar{u}, \bar{v}]^{\sigma_m}$. Moreover, $m \geq d^{\sigma}_{\max}(t)$, so $M_j[\varphi_{i+m}t', \bar{u}, \bar{v}]^{\sigma_m}$ is the unique ground term t^* such that $at^* \preccurlyeq M_j[(a\varphi_{i+m} + a)t', \bar{u}, \bar{v}]^{\sigma_m}$ and $d(t^*) > m$. Hence, if t^* is any generalised φ_n-term with $n > d^{\sigma}_{\max}(t)$ such that $t^{\sigma_m} \mapsto t^*$, then $t^* \approx M_j[\varphi_{i+m}t', \bar{u}, \bar{v}]^{\sigma_m}$.

Note that if t is an x-prefix, $m \geq d^{\sigma}_{\max}(t)$ and t^* is the generalised φ_{i+m}-term associated with t by σ_m, then t^* has no reduct with a degree in $\{m+1, m+2, \ldots, m+i-1\}$. Lemma 26 has the following consequence.

Lemma 27. *Let t be an x-prefix, let u be a y-prefix and let $m \geq \max\{d^{\sigma}_{max}(t), d^{\sigma}_{max}(u)\}$. If $\mathsf{PA}_A \vdash t^{\sigma_m} \approx u^{\sigma_m}$, then $x = y$.*

We generalise the definition of α-freeness to terms with variables: a term t is *α-free* if $t \not\approx \alpha$ and there exists no term t' such that $t \approx t' \,||\, \alpha$.

Theorem 28 (ω-completeness). *Let A be a nonempty set; then PA_A is ω-complete, i.e., for all terms t and u,*

$$\textit{if } \mathsf{PA}_A \vdash t^\sigma \approx u^\sigma \textit{ for all ground substitutions } \sigma, \textit{ then } \mathsf{PA}_A \vdash t \approx u.$$

Proof. Let $m \geq \max\{d^\sigma_{\max}(t), d^\sigma_{\max}(u)\}$. We shall prove by induction on the depth of t that if $t^{\sigma_m} \approx u^{\sigma_m}$, then $t \approx u$; clearly, this implies the theorem.

In the full version of this paper we simultaneously prove that one may assume without loss of generality that the generalised φ_n-term associated by σ_m with an x-prefix is parallel prime; we need this assumption in the remainder of the proof.

Suppose that $t^{\sigma_m} \approx u^{\sigma_m}$; it suffices to show that every summand of u is a summand of t, for then by a symmetric argument it follows that every summand of t is a summand of u, whence $t \approx u$. There are four cases:

1. If $b \in A$ such that $b \preccurlyeq u$, then $b \preccurlyeq t$ since σ_m-instances of summands of one of the three other types have a norm > 1.
2. If $a \in A$ and u' is a term such that $au' \preccurlyeq u$, then by Lemma 25 $d(u') \leq d^\sigma_{\max}(u)$. Since $m \geq d^\sigma_{\max}(u)$, $(u')^{\sigma_m}$ cannot be an immediate reduct of a σ_m-instance of an x-prefix or of a term $v \mathbin{\lfloor\!\lfloor} w$, with v an x-prefix. So there exists $i \in I$ such that $a(u')^{\sigma_m} \approx a_i t_i^{\sigma_m}$. By the induction hypothesis $u' \approx t_i$, hence $au' \preccurlyeq t$.
3. Let v be an x-prefix and let u' is a term such that $v \mathbin{\lfloor\!\lfloor} u' \preccurlyeq u$. By our assumption that generalised φ_n-terms associated by σ_m with x-prefixes are parallel prime, there exists $k \in K$ such that $v^{\sigma_m} \approx v_k^{\sigma_m}$ and $(u')^{\sigma_m} \approx u_k^{\sigma_m}$; by Lemma 27 v_k is also an x-prefix. Hence, by the induction hypothesis $v \approx v_k$ and $u' \approx u_k$, so $v \mathbin{\lfloor\!\lfloor} u \preccurlyeq t$.
4. If w is an x-prefix such that $w \preccurlyeq u$, then by our assumption that generalised φ_n-terms associated by σ_m with x-prefixes are parallel prime, there exists $l \in L$ such that $w^{\sigma_m} \approx w_l^{\sigma_m}$; by Lemma 27 w_l is also an x-prefix. If the generalised φ_n-term associated to w by σ_m is of the form $\varphi_n w'$, then clearly the generalised φ_n-term associated to w_l by σ_m must be of the form $\varphi_n w'_l$ and it is immediate by the induction hypothesis that $w' \approx w'_l$ and $w \approx w_l$. Let $w = (t' \mathbin{\lfloor\!\lfloor} u')v'$ and let $w_l = (t'' \mathbin{\lfloor\!\lfloor} u'')v''$, where t' and t'' are x-prefixes to which σ_m associates parallel prime generalised φ_n-terms $t^\dagger$ and $t^\ddagger$. If $\lceil (v')^{\sigma_m} \rceil = \lceil (v'')^{\sigma_m} \rceil$, then by the induction hypothesis $(t' \mathbin{\lfloor\!\lfloor} u') \approx (t'' \mathbin{\lfloor\!\lfloor} u'')$ and $v' \approx v''$, whence $w \approx w_l$. So let us assume without loss of generality that $\lceil (v')^{\sigma_m} \rceil < \lceil (v'')^{\sigma_m} \rceil$; then there is a ground term v^* such that $(t' \mathbin{\lfloor\!\lfloor} u')^{\sigma_m} \approx (t'' \mathbin{\lfloor\!\lfloor} u'')^{\sigma_m} v^*$ $v^*(v')^{\sigma_m} \approx (v'')^{\sigma_m}$. Note that $(t^\ddagger \mathbin{||} u'')^{\sigma_m} v^* \approx (t^\dagger \mathbin{||} u')^{\sigma_m}$, which is not parallel prime. So there exists by Proposition 23 a finite sum α of elements of A and a parallel prime generalised φ_n-term t^* such that $(t^\ddagger \mathbin{||} (u'')^{\sigma_m})v^* \approx t^* \mathbin{||} \alpha^k$. Hence by Lemma 22 $v^* \approx \alpha^l$ for some $l \geq 1$. Consequently, $v'' \approx \alpha^l v'$ and by the induction hypothesis $t' \mathbin{\lfloor\!\lfloor} u' \approx (t'' \mathbin{\lfloor\!\lfloor} u'')\alpha^l$; hence, $w \preccurlyeq t$. $\square$

References

Aceto, L. and Hennessy, M. (1993). Towards action-refinement in process algebras. *Information and Computation*, **103**(2), 204–269.

Baeten, J. C. M. and Weijland, W. P. (1990). *Process Algebra*. Number 18 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Bergstra, J. A. and Klop, J. W. (1984). Process algebra for synchronous communication. *Information and Control*, **60**(1–3), 109–137.

Castellani, I. and Hennessy, M. (1989). Distributed bisimulations. *Journal of the ACM*, **36**(4), 887–911.

Groote, J. F. (1990). A new strategy for proving ω-completeness applied to process algebra. In J. Baeten and J. Klop, editors, *Proceedings of CONCUR'90*, volume 458 of *LNCS*, pages 314–331, Amsterdam. Springer-Verlag.

Heering, J. (1986). Partial evaluation and ω-completeness of algebraic specifications. *Theoretical Computer Science*, **43**, 149–167.

Hennessy, M. (1988). Axiomatising finite concurrent processes. *SIAM Journal of Computing*, **17**(5), 997–1017.

Hirshfeld, Y. and Jerrum, M. (1998). Bisimulation equivalence is decidable for normed process algebra. LFCS Report ECS-LFCS-98-386, University of Edinburgh.

Hirshfeld, Y. and Jerrum, M. (1999). Bisimulation equivalence is decidable for normed process algebra. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *Proceedings of ICALP'99*, volume 1644 of *LNCS*, pages 412–421, Prague. Springer.

Lazrek, A., Lescanne, P., and Thiel, J.-J. (1990). Tools for proving inductive equalities, relative completeness, and ω-completeness. *Information and Computation*, **84**(1), 47–70.

Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall International, Englewood Cliffs.

Milner, R. and Moller, F. (1993). Unique decomposition of processes. *Theoretical Computer Science*, **107**, 357–363.

Moller, F. (1989). *Axioms for Concurrency*. Ph.D. thesis, University of Edinburgh.

Moller, F. (1990). The importance of the left merge operator in process algebras. In M. Paterson, editor, *Proceedings of ICALP'90*, volume 443 of *LNCS*, pages 752–764, Warwick. Springer.

A Complete Axiomatization for Observational Congruence of Prioritized Finite-State Behaviors

Mario Bravetti and Roberto Gorrieri

Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy
{bravetti, gorrieri}@cs.unibo.it

Abstract. Milner's complete proof system for observational congruence is crucially based on the possibility to equate τ divergent expressions to non-divergent ones by means of the axiom $recX.(\tau.X + E) = recX.\tau.E$. In the presence of a notion of priority, where e.g. actions of type δ have a lower priority than silent τ actions, this axiom is no longer sound because a δ action performable by E is pre-empted in the left-hand term but not in the right-hand term. The problem of axiomatizing priority using the standard observational congruence has been open for a long time. Here we show that this can be done by introducing an auxiliary operator $pri(E)$, by suitably modifying the axiom above and by introducing some new axioms. Our technique provides a complete axiomatization for Milner's observational congruence over finite-state terms of a process algebra with priority and recursion.

1 Introduction

In the last years the expressiveness of classical process algebras has been extended in several ways. Often such extensions have led to the necessity of introducing a notion of priority among actions which is useful, e.g., to model mechanisms of pre-emption. The problems arising from expressing priority have been previously studied in the context of prioritized process algebras, see e.g. [5,10,3] and [4] for a survey. One of the open questions in this context (see [4]) is finding a complete axiomatization for observational congruence in the presence of recursion.

In [4], where process algebras with priority are studied in full generality, the authors show that it is necessary to consider rather complex equivalence notions, which deviate from Milner's standard notion of observational equivalence, in order to get the congruence property. Here, we consider a less general form of priority for which the equivalences presented in [4] reduce to Milner's standard notion of observational congruence. This allows us to focus on the problem of finding a complete axiomatization for observational congruence in the presence of priority and recursion. In particular the process algebra we consider is the algebra of finite-state agents used by Milner in [9] for axiomatizing observational congruence in presence of recursion, extended with δ prefixing, where δ actions have lower priority than silent τ actions. A language like this has been studied

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 744–755, 2000.
© Springer-Verlag Berlin Heidelberg 2000

also in [6], where δ actions represent non-zero time delays, classical actions of CCS are executed in zero time, and the priority of τ actions over δ actions derives from the maximal progress assumption (i.e. the system cannot wait if it has something internal to do). As in [5,10,3,4,6] we assume that visible actions never have pre-emptive power over lower priority δ actions, because we see visible actions as indicating only the potential for execution. On the other hand, as observed e.g. in [4], in the presence of a restriction operator this assumption is necessary to get the congruence property.

In the literature complete axiomatizations of equivalences for algebras with priority have been presented for recursion free processes (e.g. in [10]). For this class of processes a notion of priority of τ actions (silent actions) over actions of a type δ is simply axiomatized by adding the axiom $\tau.E + \delta.F = \tau.E$ to the standard Milner's complete proof system for observational congruence [8,9].

When we consider also recursive processes, if we simply try to extend Milner's proof system for observational congruence [8] with the axiom above, we do not obtain a correct axiomatization. This because the axiom:

$$recX.(\tau.X + E) = recX.\tau.E \qquad (*)$$

which is part of the proof system of [9], is no longer sound. For example if $E \equiv \delta.E'$, the right-hand term can (weakly) perform δ, whilst the left-hand one cannot because of the priority of τ actions over δ actions. The axiom above is crucial for the completeness of the axiomatization in that it allows to "escape" τ divergence in weakly guarded recursive terms, during the process of turning them into strongly guarded ones. The axiom reflects the possibility of observational congruence to always equate τ divergent expressions to non-divergent ones.

The effects of priority on the possibility of escaping divergence have been previously studied in [6], where the problem of finding a complete axiomatization for recursive processes is tackled by considering a variant of Milner's standard notion of observational congruence, that makes it not always possible to escape τ divergence. In particular Milner's observational congruence is modified with the additional requirement that two bisimilar terms must have the same opportunity to silently become stable terms, i.e. terms that cannot perform τ actions. Adopting this additional constraint, which now makes observational congruence certainly sensitive to τ divergence, provides a simple necessary and sufficient condition for the possibility of escaping divergence and for applying the axiom $(*)$. Divergence can be escaped in $recX.(\tau.X + E)$ if and only if E may perform a silent move.

In this paper we show that priority and Milner's standard notion of observational congruence are compatible: we obtain a complete axiomatization for an algebra with priority and recursion even if it is always possible to escape τ divergence. This is done by replacing E in the right-hand term of axiom $(*)$ with $pri(E)$, where pri is an auxiliary operator. The behavior of $pri(E)$ is obtained from that of E by removing all transitions δ (and subsequent behaviors) performable in state E. Therefore the new axiom is:

$$recX.(\tau.X + E) = recX.\tau.pri(E)$$

Since E may be a term including free variables (e.g. in the case of nested recursions), in our axiomatization an important role in the process of turning unguarded recursive terms into strongly guarded ones is played by terms $pri(X)$. In particular we introduce the new axiom:

$$recX.(\tau.(pri(X) + E) + F) = recX.(\tau.X + E + F)$$

which is needed to remove unnecessary occurrences of terms $pri(X)$ in weakly guarded recursions. Finally we add some basic axioms that capture the δ eliminating behavior of the new auxiliary operator.

In this way we provide a complete axiomatization for observational congruence over processes of the process algebra with priority.

The paper is structured as follows. In Sect. 2 we present the algebra with priority and its operational semantics. In Sect. 3 we present a complete axiomatization for observational congruence over processes of the algebra. Finally in Sect. 4 we report some concluding remarks.

2 Prioritized Algebra

The prioritized algebra that we consider is just CCS without static operators [9] extended with δ prefixing, where δ actions have lower priority than silent τ actions. The set of prioritized actions, which includes the silent action τ denoting an internal move, is denoted by $PAct$, ranged over by $a, b, c, \ldots$. The set of all actions is defined by $Act = PAct \cup \{\delta\}$, ranged over by $\alpha, \alpha', \ldots$. The set of term variables is Var, ranged over by $X, Y, \ldots$. The set $\mathcal{E}$ of behavior expressions, ranged over by $E, F, G, \ldots$ is defined by the following syntax.

$$E ::= \underline{0} \mid X \mid \alpha.E \mid E + E \mid recX.E$$

The meaning of the operators is the standard one of [8,9], where $recX$ denotes recursion in the usual way.

As in [9] we take the liberty of identifying expressions which differ only by a change of bound variables (hence we do not need to deal with α-conversion explicitly). We will write $E\{F/X\}$ for the result of substituting F for each occurrence of X in E, renaming bound variables as necessary.

We adopt the standard definitions of [8,9] for *free* variable, and *open* and *closed* term. The set of processes, i.e. closed terms, is denoted by $\mathcal{P}$, ranged over by $P, Q, R, \ldots$.

We adopt the following standard definitions concerning guardedness of variables.

Definition 1. *X is weakly guarded in E if each occurrence of X is within some subexpression of E of the form $\alpha.F$. X is (strongly) guarded in E if we additionally require that $\alpha \neq \tau$. X is unguarded in E if X is not (strongly) guarded in E. X is fully unguarded in E if X is neither strongly nor weakly guarded in E.* ■

$$\alpha.E \xrightarrow{\alpha} E$$

$$\frac{E \xrightarrow{a} E'}{E + F \xrightarrow{a} E'} \qquad \frac{F \xrightarrow{a} F'}{E + F \xrightarrow{a} F'}$$

$$\frac{E \xrightarrow{\delta} E' \quad F \overset{\tau}{\not\longrightarrow}}{E + F \xrightarrow{\delta} E'} \qquad \frac{F \xrightarrow{\delta} F' \quad E \overset{\tau}{\not\longrightarrow}}{E + F \xrightarrow{\delta} F'}$$

$$\frac{E \xrightarrow{\alpha} E'}{recX.E \xrightarrow{\alpha} E'\{recX.E/X\}}$$

Table 1. Operational Semantics

A recursion $recX.E$ is weakly guarded, (strongly) guarded, unguarded, fully unguarded, if X is weakly guarded, (strongly) guarded, unguarded, fully unguarded in E, respectively. As in [9] we say that an expression E is guarded (unguarded) if every (some) subexpression of E which is a recursion is guarded (unguarded).

The operational semantics of the algebra terms is given as a relation $\longrightarrow \subseteq \mathcal{E} \times Act \times \mathcal{E}$. We write $E \xrightarrow{\alpha} F$ for $(E, \alpha, F) \in \longrightarrow$, $E \xrightarrow{\alpha}$ for $\exists F : (E, \alpha, F) \in \longrightarrow$ and $E \overset{\alpha}{\not\longrightarrow}$ for $\nexists F : (E, \alpha, F) \in \longrightarrow$. $\longrightarrow$ is defined as the least relation satisfying the operational rules in Table 1.

As in [10] we capture the priority of τ actions over δ actions by cutting transitions which cannot be performed directly in semantic models (and not by discarding them at the level of bisimulation definition as done in [6]) so that we can just apply the ordinary notion of observational congruence [8].

Note that the rule for recursion is not the standard one used by Milner in [8,**?**], in that it defines the moves of $recX.E$ in a pure structural way, starting from those of E (where X occurs free), instead of $E\{recX.E/X\}$. This rule, which was proven to be equivalent to that of Milner in [1], gives the possibility to simplify the proofs of some results. In particular such proofs can be made by simply inducing on the structure of terms instead of inducing on the depth of the inference of term transitions.

The equivalence notion we consider over the terms of our prioritized process algebra is the standard notion of observational congruence extended to open terms [8,**?**]. In particular we consider here the following characterization of observational congruence given in [9].

As in [9] we use $\xrightarrow{\alpha}{}^{+}$ to denote computations composed of all $\xrightarrow{\alpha}$ transitions whose length is at least one and $\xrightarrow{\alpha}{}^{*}$ to denote computations composed of all $\xrightarrow{\alpha}$ transitions whose length is possibly zero. Let $\overset{\alpha}{\Longrightarrow}$ denote

$\xrightarrow{\tau}{}^* \xrightarrow{\alpha} \xrightarrow{\tau}{}^*$. Moreover we define $\overset{\hat{\alpha}}{\Longrightarrow} = \overset{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\overset{\hat{\tau}}{\Longrightarrow} = \xrightarrow{\tau}{}^*$. Let $E \rhd X$ denote that E contains a free unguarded occurrence of X.

Definition 2. *A relation $\beta \subseteq \mathcal{E} \times \mathcal{E}$ is a weak bisimulation if, whenever $(E, F) \in \beta$:*

- *If $E \xrightarrow{\alpha} E'$ then, for some F', $F \overset{\hat{\alpha}}{\Longrightarrow} F'$ and $(E', F') \in \beta$.*
- *If $F \xrightarrow{\alpha} F'$ then, for some E', $E \overset{\hat{\alpha}}{\Longrightarrow} E'$ and $(E', F') \in \beta$.*
- *$E \rhd X$ iff $F \rhd X$.*

Two expressions E, F are weakly bisimilar, written $E \approx F$, iff (E, F) is included in some weak bisimulation. ■

Definition 3. *Two expressions E, F are observational congruent, written $E \simeq F$, iff:*

- *If $E \xrightarrow{\alpha} E'$ then, for some F', $F \overset{\alpha}{\Longrightarrow} F'$ and $E' \approx F'$.*
- *If $F \xrightarrow{\alpha} F'$ then, for some E', $E \overset{\alpha}{\Longrightarrow} E'$ and $E' \approx F'$.*
- *$E \rhd X$ iff $F \rhd X$.* ■

Corollary 4. *If $E \simeq F$ then:*

$$E \xrightarrow{\tau} \quad \Leftrightarrow \quad F \xrightarrow{\tau}$$ ■

The following theorem shows that the presence of priority preserves the congruence property of observational congruence w.r.t. the operators of the algebra.

Theorem 5. *$\simeq$ is a congruence w.r.t. prefix, choice and recursion operators.*

Proof In the case of the prefix operator the proof is trivial. As far as the choice operator is concerned, the only problematic case arises when, given $E \simeq F$, we have a move $E + G \xrightarrow{\delta} H$ derived from $G \xrightarrow{\delta} H$. In this case we can conclude that $F + G \xrightarrow{\delta} H$ by using Corollary 4. See [2] for a detailed proof which also includes the case of the recursion operator. ■

3 Axiomatization

We now present an axiomatization of $\simeq$ which is complete over processes $P \in \mathcal{P}$ of our algebra.

As we already explained in the introduction, the law of ordinary CCS which allows to escape τ divergence:

$$recX.(\tau.X + E) = recX.\tau.E$$

is not sound in a calculus with this kind of priority. Consider for instance the divergent term $F \equiv recX.(\tau.X + \delta.\underline{0})$. Because of priority of "τ" actions over "δ" actions the operational semantics of F is isomorphic to that of $recX.\tau.X$.

Hence F is an infinitely looping term which can never escape from divergence by executing the action "δ". If the cited law were sound, we would obtain $F = recX.\tau.\delta.\underline{0}$ and this is certainly not the case.

In order to overcome this problem in [6] the distinguished symbol "$\perp$" is introduced, which represents an ill-defined term that can be removed from a summation only if a silent computation is possible. In this way by considering the rule $recX.(\tau.X + E) = recX.\tau.(E + \perp)$ the resulting term which escapes divergence can be turned into a "normal" term only if E may execute a silent move.

This law is surely sound (over terms without "$\perp$") also in our language, but is not sufficient to achieve completeness. Since, differently from [6], we do not impose conditions about stability in our definition of observational congruence, we can escape divergence not only when E includes a silent move but *for all possible* terms E. For example in our calculus (but not in [6]) the term $recX.\tau.X$ is equivalent to $\tau.\underline{0}$ (as in standard CCS), so we can escape divergence in F even if $\tau.X$ has not a silent alternative inside $recX$. In general the behavior of E' such that $recX.\tau.E' = recX.(\tau.X + E)$ is obtained from that of E by removing all "δ" actions (and subsequent behaviors) performable in state E. We denote such E', representing the "prioritized" behavior of state E, with $pri(E)$. The operational semantics of the auxiliary operator pri is just:

$$\frac{E \xrightarrow{\alpha} E'}{pri(E) \xrightarrow{\alpha} E'} \qquad \alpha \neq \delta$$

Therefore in our case τ divergence can always be escaped by turning E into $pri(E)$ and the strongly guarded terms we obtain are always "well-defined" terms.

Note that the introduction of this auxiliary operator is crucial for being able to axiomatize the priority of τ actions over δ actions in our case. Since we have to remove δ actions performable by a term E even if E does not include a silent move, we cannot do this by employing a special symbol like "$\perp$" instead of using an operator. This because $\perp$ must somehow be removed at the end of the deletion process (in [6] $\perp$ is eliminated by silent alternatives) in order to obtain a "normal" term.

The axiomatization of "$\simeq$" we propose is made over the set of terms $\mathcal{E}_{pri}$, generated by extending the syntax to include the new operator $pri(E)$. We start by noting that the congruence property trivially extends to this operator.

Theorem 6. *$\simeq$ is a congruence w.r.t. the new operator $pri(E)$.*

We adopt the following notion of *serial* variable, which is used in the axiomatization.

Definition 7. *X is serial in $E \in \mathcal{E}_{pri}$ if every subexpression of E which contains X, apart from X itself, is of the form $\alpha.F$, $F' + F''$ or $recY.F$ for some variable Y.* ■

$(A1)$	$E + F = F + E$
$(A2)$	$(E + F) + G = E + (F + G)$
$(A3)$	$E + E = E$
$(A4)$	$E + \underline{0} = E$
$(Tau1)$	$\alpha.\tau.E = \alpha.E$
$(Tau2)$	$E + \tau.E = \tau.E$
$(Tau3)$	$\alpha.(E + \tau.F) + \alpha.F = \alpha.(E + \tau.F)$
$(Pri1)$	$pri(\underline{0}) = \underline{0}$
$(Pri2)$	$pri(a.E) = a.E$
$(Pri3)$	$pri(\delta.E) = \underline{0}$
$(Pri4)$	$pri(E + F) = pri(E) + pri(F)$
$(Pri5)$	$pri(pri(E)) = pri(E)$
$(Pri6)$	$\tau.E + F = \tau.E + pri(F)$
$(Rec1)$	$recX.E = E\{recX.E/X\}$
$(Rec2)$	$F = E\{F/X\} \Rightarrow F = recX.E$ provided that X is strongly guarded and serial in E
$(Ung1)$	$recX.(X + E) = recX.E$
$(Ung2)$	$recX.(\tau.X + E) = recX.(\tau.pri(E))$
$(Ung3)$	$recX.(\tau.(X + E) + F) = recX.(\tau.X + E + F)$
$(Ung4)$	$recX.(\tau.(pri(X) + E) + F) = recX.(\tau.X + E + F)$

Table 2. Axiom system $\mathcal{A}$

The axiom system $\mathcal{A}$ is formed by the axioms presented in Table 2.

The axiom $(Pri6)$ expresses the priority of τ actions over δ actions. Note that from $(Pri6)$ we can derive $\tau.E + \delta.E = \tau.E$ by applying $(Pri3)$. The axioms $(Rec1)$, $(Rec2)$ handle strongly guarded recursion in the standard way [9]. The axioms $(Ung1)$ and $(Ung2)$ are used to turn unguarded terms into strongly guarded ones similarly as in [9]. The axiom $(Ung3)$ and the new axiom $(Ung4)$ are used to transform weakly guarded recursions into the form required by the axiom $(Ung2)$, so that they can be turned into strongly guarded ones. In particular the role of axiom $(Ung4)$ is to remove unnecessary occurrences of terms $pri(X)$ in weakly guarded recursions.

3.1 Soundness

Theorem 8. *Given $E, F \in \mathcal{E}_{pri}$, if $\mathcal{A} \vdash E = F$ then $E \simeq F$.*

Proof The soundness of the laws $(Rec1)$ and $(Rec2)$ derives from the fact that systems of equations of the form $X_i = E_i$, where the variables X_i are guarded and serial in terms E_i and are not in the scope of any recursion, have a unique solution up to provable equality. The proof of this fact is a straightforward adaptation of the proof given in [8]. The soundness of the new equations $(Pri1) - (Pri6)$ trivially derives from the fact that the semantic models of left-hand and right-hand terms are isomorphic. The soundness of the laws $(Ung1) - (Ung4)$ is proved in [2]. ■

3.2 Completeness

Now we will show that the axiom system $\mathcal{A}$ is complete over processes of $\mathcal{P}$. In order to do this we follow the lines of [9], so we deal with systems of recursive equations.

We start by introducing the machinery necessary for proving completeness over guarded expressions $E \in \mathcal{E}$. Afterwards we will show how the axioms (Ung) can be used to turn an unguarded processes of $\mathcal{P}$ into a guarded process of $\mathcal{P}$. Note that the new operator $pri(E)$ is used only in the intermediate steps of the second procedure.

Definition 9. *An equation set with formal variables $\tilde{X} = \{X_1, \ldots, X_n\}$ and free variables $\tilde{W} = \{W_1, \ldots, W_m\}$, where $\tilde{X}$ and $\tilde{W}$ are disjoint, is a set $S = \{X_i = H_i \mid 1 \leq i \leq n\}$ of equations such that the expressions H_i $(1 \leq i \leq n)$ have free variables in $\tilde{X} \cup \tilde{W}$.* ■

We say that an expression E *provably satisfies* S if there are expressions $\tilde{E} = \{E_1, \ldots, E_n\}$ with free variables in $\tilde{W}$ such that $E_1 \equiv E$ and for $1 \leq i \leq n$ we have $\mathcal{A} \vdash E_i = H_i\{\tilde{E}/\tilde{X}\}$.

The equation sets that are actually dealt with in the proof of completeness of [9] belong to the subclass of *standard equation sets*. Here we have to slightly change the characterization of this subclass because of the presence of priority.

Definition 10. *An equation set $S = \{X_i = H_i \mid 1 \leq i \leq n\}$, with formal variables $\tilde{X} = \{X_1, \ldots, X_n\}$ and free variables $\tilde{W} = \{W_1, \ldots, W_m\}$, is standard if each expression H_i $(1 \leq i \leq n)$ is of the form:* [1]

$$H_i \equiv \sum_{j \in J_i} \alpha_{i,j}.X_{f(i,j)} + \sum_{k \in K_i} W_{g(i,k)}$$

where:

$$\exists j \in J_i : \alpha_{i,j} = \tau \;\Rightarrow\; \nexists j \in J_i : \alpha_{i,j} = \delta \;.$$

■

As in [9], for a standard equation set S we define the relations $\longrightarrow_S \subseteq \tilde{X} \times Act \times \tilde{X}$ and $\rhd_S \subseteq \tilde{X} \times \tilde{W}$ as follows:

$$\begin{array}{lll} X_i \xrightarrow{\alpha}_S X & \text{iff} & \alpha.X \text{ occurs in } H_i \\ X_i \rhd_S W & \text{iff} & W \text{ occurs in } H_i \end{array}$$

[1] We assume $\sum_{j \in J} E \equiv \underline{0}$ if $J = \emptyset$.

Definition 11. *A standard equation set S with formal variables $\tilde{X} = \{X_1, \ldots, X_n\}$ is guarded if there is no cycle $X_i \xrightarrow{\tau}_S^+ X_i$.* ■

The following theorem guarantees that from a guarded expression $E \in \mathcal{E}$ we can derive a standard guarded equation set which is provably satisfied by E.

Theorem 12. *Every guarded expression $E \in \mathcal{E}$ provably satisfies a standard guarded equation set S.*

Proof The proof is as in [9], where, in addition, we modify the resulting equation set by eliminating the occurrences of $\delta.X$ (for some X) in the equations which include terms $\tau.Y$ (for some Y). This is done by using laws $(Pri6)$ and $(Pri3)$. ■

Once established the form of standard guarded equation sets S, completeness over guarded expressions is obtained by saturating equation sets S as in [9]. In particular the proofs of the following lemma and two theorems are the same as in [9].

Definition 13. *A standard equation set S with formal variables $\tilde{X}$ is saturated if, for all $X \in \tilde{X}$:*

$$\begin{array}{llll} (i) & X \xrightarrow{\tau}_S^* \xrightarrow{\alpha}_S \xrightarrow{\tau}_S^* X' & \Rightarrow & X \xrightarrow{\alpha} X' \\ (ii) & X \xrightarrow{\tau}_S^* \rhd_S W & \Rightarrow & X \rhd_S W \end{array}$$

■

Lemma 14. *Let $E \in \mathcal{E}$ provably satisfy S, standard and guarded. Then there is a saturated, standard and guarded equation set S' provably satisfied by E.* ■

The possibility of saturating standard and guarded equation sets S leads to the following theorem.

Theorem 15. *Let $E \in \mathcal{E}$ provably satisfy S, and $F \in \mathcal{E}$ provably satisfy T, where both S and T are standard, guarded sets of equations, and let $E \simeq F$. Then there is a standard, guarded equation set U provably satisfied by both S and T.* ■

Theorem 16. *Let $E, F \in \mathcal{E}$ provably satisfy the same standard guarded equation set S, then $\mathcal{A} \vdash E = F$.* ■

Hence we have proved completeness over guarded expressions.

Theorem 17. *If E and F are guarded expressions of $\mathcal{E}$ and $E \simeq F$ then $\mathcal{A} \vdash E = F$.* ■

Now we show that each unguarded process can be turned into a guarded process of $\mathcal{P}$, so that we obtain completeness also over unguarded processes. We start with a technical lemma which, in analogy to [9], is important to obtain this result.

Lemma 18. *If X occurs free and unguarded in $E \in \mathcal{E}$, then $\mathcal{A} \vdash E = X + E$.* ■

The proof of this lemma is the same as in [9].

Theorem 19. *For each process $P \in \mathcal{P}$ there exists a guarded $P' \in \mathcal{P}$ such that $\mathcal{A} \vdash P = P'$.*

Proof We show, by structural induction, that given an expression $E \in \mathcal{E}$, it is possible to find an expression $F \in \mathcal{E}_{pri}$ such that:

1. if $pri(G)$ is a subexpression of F then $G \equiv X$ for some free variable X;
2. for any variable X, $pri(X)$ is weakly guarded in F, i.e. each occurrence of $pri(X)$ is within some subexpression of F of the form $\alpha.G$;
3. a summation cannot have both $pri(X)$ and Y as arguments, for any (possibly coincident) variables X and Y;
4. for any variable X, each subterm $recX.G$ of F is (strongly) guarded in F, i.e. each occurrence of $recX.G$ is within some subexpression of F of the form $\alpha.H$, with $\alpha \neq \tau$;
5. F is guarded;
6. $\mathcal{A} \vdash E = F$.

Note that a consequence of property 4 is that each unguarded occurrence of any free variable X of F does not lie within the scope of a subexpression $recY.G$ of F.

Showing this result proves the theorem, in that if $E \in \mathcal{E}$ is a process of $\mathcal{P}$, i.e. a closed term, we have (by the properties of F above) that F is also a process of $\mathcal{P}$, it is guarded, and $\mathcal{A} \vdash E = F$. The result above is derived by structural induction on the syntax of an expression $E \in \mathcal{E}$. In particular in the case $E \equiv recX.E'$ a fundamental role is played by Lemma 18 which generates from a term E'' such that X $(pri(X))$ is unguarded in E'' an equivalent term $X + E''$ $(pri(X) + E'')$ so that law $(Ung3)$ (law $(Ung4)$) can be applied. See [2] for a complete proof. ■

From Theorem 17 and Theorem 19 we derive the completeness of $\mathcal{A}$ over processes of $\mathcal{P}$.

Theorem 20. *Given $P, Q \in \mathcal{P}$, if $P \simeq Q$ then $\mathcal{A} \vdash P = Q$.* ■

Note that all the axioms of $\mathcal{A}$ are actually used in the proof of completeness. In particular in the proof of completeness over guarded expressions (Theorem 17) we employ the standard axioms $(A1) - (A4)$, $(Tau1) - (Tau3)$ and $(Rec1), (Rec2)$ as in [9], plus the new axioms $(Pri3)$ and $(Pri6)$. All these axioms are necessary even if we restrict to consider completeness over guarded processes only. Moreover, proving that a process of $\mathcal{P}$ can always be turned into a guarded process (Theorem 19) requires the use of the remaining axioms $(Pri1), (Pri2), (Pri4), (Pri5)$ and $(Ung1) - (Ung4)$ [2]. This supports the claim that our axiomatization is irredundant.

4 Conclusion

The algebra presented in this paper and its axiomatization can be extended to include all the operators of CCS, by employing operational rules like those presented in [4]. This can be done easily if parallel composition is assumed to have an operational semantics implementing *local* pre-emption (and not *global* pre-emption, see [4]). This means that τ actions of a sequential process may pre-empt only actions δ of the same sequential process. For instance in $\tau.E|\delta.F$ the action δ of the righthand process is not pre-empted by the action τ of the lefthand process, as instead happens if we assume global pre-emption. On the other hand, local pre-emption seems to be natural in the context of distributed systems, where the choices of a process do not influence the choices of another process.

If global pre-emption is, instead, assumed, then Milner's standard notion of observational equivalence is not a congruence for the parallel operator (see [4]) with the usual operational rule, where $E|F$ may perform a δ action (originating from E or F) only if it cannot execute any τ action. This because, e.g., $recX.\tau.X$ is observationally congruent to $\tau.\underline{0}$, but $recX.\tau.X|\delta.\underline{0}$, whose semantics, due to global pre-emption, is that of $recX.\tau.X$, is not observationally congruent to $\tau.\underline{0}|\delta.\underline{0}$, whose semantics is that of $\tau.\delta.\underline{0}$. In this case a possibility is to resort to a finer notion of observational congruence similar to that presented in [6].

On the other hand, when global priority derives from execution times associated with actions as in [6], where δ actions represent non-zero time delays and classical actions of CCS are executed in zero time, the operational rule for the parallel operator implementing global pre-emption of [4] does not seem the most natural one. In this context a process like $recX.\tau.X$ represents a Zeno process which executes infinite τ actions in the same time point. According to the operational rule for parallel operator of [4], the semantic model of $recX.\tau.X|\delta.\underline{0}$, as already stated, is that of $recX.\tau.X$, where we observe only the τ moves of the lefthand Zeno process which are all executed in zero time. This is in contrast with the intuition that in $recX.\tau.X|\delta.\underline{0}$ the righthand process should eventually advance because, since no synchronization occurs between the two processes, the behavior of the righthand process should not be influenced from that of the lefthand process. The point is that the infinite sequence of τ moves executed by the lefthand process prevents the time from advancing and as a consequence, by employing the operational rule of [4], we do not observe the behavior of the system after the time point where the τ actions are executed. This leads to an incomplete description of the behavior of $recX.\tau.X|\delta.\underline{0}$ that makes the semantics of $recX.\tau.X|\delta.\underline{0}$ to be different from that of $\tau.\underline{0}|\delta.\underline{0}$. Therefore it seems that the real point here is not to consider a finer notion of observational congruence that makes $recX.\tau.X$ not equivalent to $\tau.\underline{0}$ (as done in [6]), but to have an operational semantics for the parallel operator that allows to observe the behavior of the system after the critical time point. The definition of such a semantics is an open problem which we are currently working on.

Acknowledgements

We thank the anonymous referees for their helpful comments. This research has been partially funded by MURST progetto TOSCA.

References

1. E. Badouel, P. Darondeau, *"On Guarded Recursion"*, in Theoretical Computer Science 82(2):403-408, 1991
2. M. Bravetti, R. Gorrieri, *"A Complete Axiomatization for Observational Congruence of Prioritized Finite-State Behaviors"*, Technical Report UBLCS-99-18, University of Bologna (Italy), 1999
3. R. Cleaveland, G. Luttgen, V. Natarajan, *"A Process Algebra with Distributed Priorities"*, in Proc. of the *7th Int. Conf. on Concurrency Theory (CONCUR '96)*, LNCS 1119:34-49, 1996
4. R. Cleaveland, G. Luttgen, V. Natarajan, *"Priority in Process Algebras"*, to appear in Handbook of Process Algebra, Elsevier, 2000
5. R. Cleaveland, M. Hennessy, *"Priorities in Process Algebra"*, in Information and Computation 87:58-77, 1990
6. H. Hermanns, M. Lohrey, *"Priority and Maximal Progress Are Completely Axiomatisable (Extended Abstract)"*, in Proc. of the *9th Int. Conf. on Concurrency Theory (CONCUR '98)*, LNCS 1466:237-252, Nice (France), 1998
7. H. Hermanns, M. Lohrey, *"Observational Congruence in a Stochastic Timed Calculus with Maximal Progress"*, Technical Report IMMD-VII/7-97, Universität Erlangen-Nürnberg, 1997
8. R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989.
9. R. Milner, *"A complete axiomatization for observational congruence of finite-state behaviours"*, in Information and Computation 81:227-247, 1989
10. V. Natarajan, I. Christoff, L. Christoff, R. Cleaveland, *"Priorities and Abstraction in Process Algebra"*, in Proc. of *Foundations of Software Technology and Theoretical Computer Science*, LNCS 880:217-230, 1994

Tight Size Bounds for Packet Headers in Narrow Meshes *

Micah Adler[1], Faith Fich[2], Leslie Ann Goldberg[3], and Mike Paterson[3]

[1] Department of Computer Science, University of Massachusetts, Amherst, MA 01003, USA. micah@cs.umass.edu
[2] Department of Computer Science, University of Toronto, Toronto, Canada, M5S 3G4.† fich@cs.toronto.edu
[3] Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK.‡ {leslie,msp}@dcs.warwick.ac.uk

Abstract. Consider the problem of sending a single message from a sender to a receiver through an $m \times n$ mesh with asynchronous links that may stop working, and memoryless intermediate nodes. We prove that for $m \in O(1)$, it is necessary and sufficient to use packet headers that are $\Theta(\log \log n)$ bits long.

1 Introduction

Protocols that send information bundled into packets over a communication network allocate some number of bits in each packet for transmitting control information. We here refer to such bits as *header bits*. These bits might include sequence numbers to ensure that packets are received in the correct order, or they might contain routing information to ensure that a packet is delivered to its destination. When the number of message bits in a packet is small (for example, in acknowledgements), the header bits can make up a significant fraction of the total number of bits contained in the packet. A natural question to ask is the following: how large must packet headers be for reliable communication?

This problem is addressed in [AF99], part of a large body of research on the end-to-end communication problem [AAF+94], [AAG+97], [AG88], [AMS89], [APV96], [DW97], [KOR95], [LLT98], [F98]. The *end-to-end communication* problem is to send information from one designated processor (the *sender* S) to another designated processor (the *receiver* R) over an unreliable communication network. This is a fundamental problem in distributed computing, since (a) communication is crucial to distributed computing and (b) as the size of a network increases, the likelihood of a fault occurring somewhere in the network also increases.

* A full version appears at http://www.dcs.warwick.ac.uk/~leslie/papers/endtoend.ps.

† This work was partly supported by grants from NSERC and CITO.

‡ This work was partly supported by EPSRC grant GR/L60982 and ESPRIT LTR Project ALCOM-FT.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 756–767, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Adler and Fich [AF99] studied the question of how many header bits are required for end-to-end communication in the setting where links may fail. They prove that, for the complete network of n processors or any network that contains it as a minor (such as the n^2-input butterfly or the $n \times n \times 2$ mesh), any memoryless protocol that ensures delivery of a single message using headers with fewer than $\lceil \log_2 n \rceil - 3$ bits, generates an infinite amount of message traffic.

If there is a path of live links from S to R in an n-node network, then there is a simple path of live links of length at most $n - 1$. Therefore, it suffices to use the simple "hop count" algorithm [P81] which discards messages that have been forwarded $n - 1$ times. Since this can be done with headers of size $\lceil \log n \rceil$, for the complete graph we have upper and lower bounds that match to within a small additive constant, and for the n^2-input butterfly and the $n \times n \times 2$ mesh to within a small multiplicative constant.

However, for several graphs there remains a large gap between the best upper and lower bounds. Planar graphs, including two-dimensional meshes, do not contain a complete graph on more than 4 nodes as a minor [K30] and, as a result, no previous work has demonstrated a lower bound larger than a constant for any planar graph. Furthermore, for some graphs it is possible to do better than the simple hop count algorithm. For example, Adler and Fich [AF99] observed that if F is a *feedback vertex set* of the underlying graph G (that is, if every cycle of G contains at least one vertex of F), then one can use a variant of the hop count protocol which discards messages that have visited F more than $|F|$ times. The discarding does no harm, since a simple path visits F at most $|F|$ times. But it ensures that the amount of traffic generated is finite. Note that in this variant of the hop count protocol, the length of packet headers is at most $\lceil \log_2(|F| + 1) \rceil$.

However, some graphs have no small feedback vertex sets. In particular, any feedback vertex set for the $m \times n$ mesh has size at least $\lfloor m/2 \rfloor \cdot \lfloor n/2 \rfloor$. In this case, this variant does not offer significant improvement over the hop count algorithm.

Thus we see that a network that has resisted both lower bound and upper bound improvements is the two-dimensional mesh. Prior to this work, there was no upper bound better than $O(\log mn)$, nor lower bound better than $\Omega(1)$, for any $m \times n$ mesh with $m, n > 2$. For $m = 2$, headers of length one suffice in our network (since no backward move is needed and we need only distinguish vertical and horizontal arrivals) [AF99]. In [AF99], it is conjectured that $\Omega(\log n)$ header bits are necessary for a protocol to ensure delivery of a single message in an $n \times n$ mesh without generating an infinite amount of message traffic.

Here, we attack this open problem by considering $m \times n$ meshes, for constant $m \geq 3$. We prove the unexpected result that $\Theta(\log \log n)$ bit headers are necessary and sufficient for such graphs.

1.1 Network Model

We model a network by an undirected graph G, with a node corresponding to each processor and an edge corresponding to a link between two processors. Specifically, we consider the graphs $G(m, n)$ with a sender node S and a receiver

node R in addition to the mn intermediate nodes, (i, j), for $0 \leq i < m$ and $0 \leq j < n$. There are links between

- node S and node $(i, 0)$, for $0 \leq i < m$,
- node (i, j) and node $(i, j+1)$, for $0 \leq i < m$ and $0 \leq j < n-1$,
- node (i, j) and node $(i+1, j)$, for $0 \leq i < m-1$ and $0 \leq j < n$, and
- node $(i, n-1)$ and node R, for $0 \leq i < m$.

The graph $G(3, 6)$ is illustrated in Figure 1.

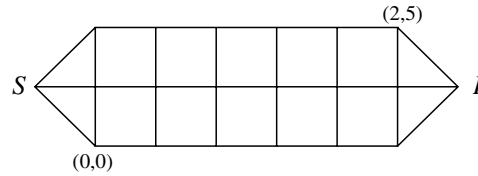


Fig. 1. The graph $G(3, 6)$

Processors communicate by sending packets along links in the network. Each packet consists of data (i.e. the message) and a header. The processor at an intermediate node may use information in the header to determine what packets to send to its neighbours, but they cannot use the data for this purpose. Headers may be modified arbitrarily; however, data must be treated as a "black box". That is, processors may make copies of the data, but they may not modify it. This *data-oblivious* assumption is appropriate when one views end-to-end communication protocols as providing a reliable communication layer that will be used by many different distributed algorithms. Typically in end-to-end communication, one assumes that when a processor receives a packet, it cannot detect which of its neighbours sent the packet. This assumption is not relevant to our problem since the degree of the underlying network is bounded, and so the identity of the sender can be encoded in a constant number of header bits.

Intermediate processors are assumed to be *memoryless*. Thus, processors can only send packets as a result of receiving a packet and must decide along which link(s) to forward the message and how to change the packet header, based only on the contents of the header. This is an appropriate model for a network with simultaneous traffic between many different pairs of processors, for example, the Internet, where no information concerning past traffic is stored.

The links of the network are either *alive* or *dead*. At any time, a live link may become dead. Once a link becomes dead, it remains so. Processors do not know which subset of the links are alive. For simplicity, it is assumed that processors never fail. However, a dead processor can be simulated by considering each of its incident links to be dead.

Live links deliver packets in a first in, first out manner. However, the time for a packet to traverse a link may differ at different times or for different links.

We assume that the time for a packet to traverse a link is finite, but unbounded. Edges which are dead can be thought of as having infinite delay. In this asynchronous model, a processor cannot distinguish between a dead link and a link which is just very slow.

1.2 Summary of Results

In this paper, we consider the problem of sending a single message from S to R. Our goal is to ensure that

- as long as there is some simple S–R path of live links, at least one copy of the message gets sent from S to R, and
- even if all links are alive, only a finite number of packets are generated.

We say that a protocol which satisfies these requirements *delivers a message from S to R with finite traffic*. In this paper, we provide an algorithm that does this using $O(m(\log\log n + \log m))$-bit headers for any network $G(m,n)$. For the case of $G(3,n)$, this is improved in the full version of our paper to $\log_2\log_2 n + O(1)$. In Section 3 we demonstrate that for $G(3,n)$, $\log_2\log_2 n - O(\log\log\log n)$ bits are required. Using the following observation of Adler and Fich [AF99], this lower bound can be extended to $G(m,n)$.

Proposition 1. *Suppose G' is a minor of G and S' and R' are the supernodes of G' containing S and R, respectively. Then any protocol for G that delivers a message from S to R with finite traffic gives a protocol for G' with the same packet headers that delivers a message from S' to R' with finite traffic.*

In particular, since for $m \geq 3$, $G(m,n)$ has $G(3,n)$ as a minor, $\log_2\log_2 n - O(\log\log\log n)$ bits are required for $G(m,n)$. Thus, for any constant $m \geq 3$, we have optimal bounds to within a constant factor on the number of header bits that are necessary and sufficient to deliver a message from S to R with finite traffic in $G(m,n)$. For the case of $G(3,n)$, our bounds are within an additive term of $O(\log\log\log n)$ from optimal.

Our upper bounds use a new technique to obtain an approximate count of how many nodes a message has visited, which is sufficient to guarantee that only a finite number of packets are generated. This technique may have applications to other networks. By Proposition 1, our upper bounds also provide upper bounds for any graphs that are minors of $G(m,n)$, for any constant m.

The next section describes our protocol for $G(m,n)$ for any constant $m \geq 3$. This is followed in Section 3 by our lower bound for $G(3,n)$ and, hence, for $G(m,n)$ with $m > 3$.

2 A Protocol for $G(m,n)$

In this section, we provide an upper bound on the header size required for sending a single message from S to R in $G(m,n)$. Since $G(m,n)$ is a minor of $G(m,n')$

for all $n \leq n'$, by Proposition 1, it suffices to assume that $n = 2^h + 1$ for some positive integer h.

We begin by giving a characterization of certain simple paths. The characterization will be used in Lemma 1 to parse simple paths. We will be considering sub-paths that go from left-to-right (from small c_1 to big c_2) and also sub-paths that go from right-to-left (from big c_1 to small c_2), but we will always work within a bounded region of rows consisting of row r_1 up to row r_2.

Definition 1. *For $r_1 \leq r_2$ and $c_1 \neq c_2$, a* (c_1, c_2, r_1, r_2)-bounded path *is a simple path that starts in column c_1, ends in column c_2, and does not go through any node in a column less than* $\min\{c_1, c_2\}$*, a column greater than* $\max\{c_1, c_2\}$*, a row less than r_1, or a row greater than r_2.*

Note that every simple path from the first column of $G(m, n)$ to the last column of $G(m, n)$ is a $(0, n-1, 0, m-1)$-bounded path. A (c_1, c_2, r, r)-bounded path is a simple path of horizontal edges.

Definition 2. *For $r_1 < r_2$ and $c_1 \neq c_2$, a* (c_1, c_2, r_1, r_2)-bounded loop *is a simple path that starts and ends in column c_1, and does not go through any node in a column less than* $\min\{c_1, c_2\}$*, a column greater than* $\max\{c_1, c_2\}$*, a row less than r_1, or a row greater than r_2.*

We focus attention on bounded paths between columns which are consecutive multiples of some power of 2, i.e. from column $c2^k$ to column $c'2^k$, where $c' = c \pm 1$.

Lemma 1. *Let c_1, c_2, c_3 be consecutive nonnegative integers, with c_2 odd, and let k be a nonnegative integer. Then every $(c_1 2^k, c_3 2^k, r_1, r_2)$-bounded path can be decomposed into a $(c_1 2^k, c_2 2^k, r_1, r_2)$-bounded path, followed by a series of $r_2 - r_1$ or fewer $(c_2 2^k, c_1 2^k, r_1, r_2)$- and $(c_2 2^k, c_3 2^k, r_1, r_2)$-bounded loops, followed by a $(c_2 2^k, c_3 2^k, r_1, r_2)$-bounded path.*

Proof. Consider any $(c_1 2^k, c_3 2^k, r_1, r_2)$-bounded path. The portion of the path until a node in column $c_2 2^k$ is first encountered is the first subpath, the portion of the path after a node in column $c_2 2^k$ is last encountered is the last subpath, and the remainder of the path is the series of loops starting and ending in column $c_2 2^k$. The bound on the number of loops follows from the fact that the path is simple, so the first subpath and each of the loops end on different nodes in column $c_2 2^k$. □

This gives us a recursive decomposition of any simple path from the first column to the last column of $G(m, n)$, where n is one more than a power of 2. Specifically, such a $(0, n-1, 0, m-1)$-bounded path consists of a $(0, (n-1)/2, 0, m-1)$-bounded path, followed by a series of at most $m-1$ different $((n-1)/2, n-1, 0, m-1)$ and $((n-1)/2, 0, 0, m-1)$-bounded loops, followed by a $((n-1)/2, n-1, 0, m-1)$-bounded path. Each of the bounded paths can then be similarly decomposed. Furthermore, we can also decompose the bounded loops.

Lemma 2. *Let k, r_1, r_2, c_1 and c_2 be nonnegative integers, where c_1 and c_2 are consecutive, c_1 is odd, and $r_1 < r_2$. Then every $(c_1 2^k, c_2 2^k, r_1, r_2)$-bounded loop can be decomposed into the prefix of a $(c_1 2^k, c_2 2^k, r_1 + 1, r_2)$-bounded path, followed by a downward edge, followed by the suffix of a $(c_2 2^k, c_1 2^k, r_1, r_2 - 1)$-bounded path, or the prefix of a $(c_1 2^k, c_2 2^k, r_1, r_2 - 1)$-bounded path, followed by an upward edge, followed by the suffix of a $(c_2 2^k, c_1 2^k, r_1 + 1, r_2)$-bounded path.*

Proof. Consider any $(c_1 2^k, c_2 2^k, r_1, r_2)$ bounded loop. Let c be the column farthest from $c_1 2^k$ that this path reaches and let (r, c) be the first node in this path in column c. Let p_1 be the prefix of this path up to and including node (r, c). The next edge is vertical. Let p_2 be the remainder of the bounded loop following that edge.

Since the loop is a simple path, paths p_1 and p_2 do not intersect. Thus, either p_1 is completely above p_2, so p_1 never uses row r_1 and p_2 never uses row r_2, or p_1 is completely below p_2, so p_1 never uses row r_2 and p_2 never uses row r_1. □

We use this recursive decomposition of simple paths in our protocol. Instead of trying just the simple S–R paths in $G(m, n)$, our protocol tries **all** S–R paths that can be recursively decomposed in this way.

Our basic building block is a protocol that sends a packet from column c_1 to column c_2, where c_1 and c_2 are consecutive multiples of some power of 2, using some set of r adjacent rows. The protocol does this by first sending the packet from column c_1 to the middle column $(c_1 + c_2)/2$, recursively. Then it sends the packet looping around the middle column at most $r - 1$ times. Each loop consists of a first half and a second half, each of which uses at most $r - 1$ rows. Both of these subproblems are solved recursively. Finally, the protocol recursively sends the packet from the middle column to column c_2.

It follows by Lemmas 1 and 2 that, if there is a simple path of live edges from S to R, then our protocol finds it. Note that, at the lowest level of the recursion, a packet is always travelling in what is considered the forward direction (when the bounded path is from right to left, this will be in the backwards direction of the original problem, but still in the forward direction of the lowest level subproblem). Thus, the difficult part of this protocol is performing the bounded loops in such a way that the packet does not travel in an infinite loop.

Let $\#_2(0) = \infty$ and for every positive integer c, let $\#_2(c)$ denote the largest power of two that divides c. Thus, if c can be expressed as $c_1 2^k$ for an odd number c_1, then $\#_2(c) = k$. In our protocol, the packet header is used to keep track of the column in which the current loop started and the distance to the other column boundary. If we naively stored these numbers, then $\Omega(\log n)$ header bits would be required. However, because our decomposition only uses bounded loops of the form $(c_1 2^k, (c_1 \pm 1)2^k, r_1, r_2)$, where c_1 is odd, it is sufficient to keep track of k (i.e., $\#_2(c_1 2^k)$). Note that k can be represented using only $\lceil \log_2 \log_2 (n - 1) \rceil$ bits. Using the quantity k, a packet can tell when it reaches its boundary columns. In particular, while its current column c is *between* the boundaries, $\#_2(c) < k$ but when c is at the boundaries $\#_2(c) \geq k$.

When the algorithm is doing a bounded loop from column $c_1 2^k$ the following quantities are stored.

- *power* $= \#_2(c_1 2^k)$ (which is equal to k),
- *minRow*, the smallest row that can be used,
- *maxRow*, the largest row that can be used,
- *loopCounter*, the number of loops that have already been done around column $c_1 2^k$ in the current path,
- *loopHalf* (0 if the current packet is in the first bounded path that forms this loop and +1 if it is in the second),
- *forward*, the direction in which the packet is travelling on the current path (+1 if the packet is going from left to right and −1 it is going from right to left).

Although our path decomposition has $\log_2(n-1)$ levels of recursion, at most m loops can be active at any one time. This follows from Lemma 2, since the number of allowed rows decreases by 1 for each active loop. We shall think of the bits in the packet header as a stack and, for each active loop, the above mentioned variables will be pushed onto the stack. Finally, we use two additional bits with each transmission to ensure that any node receiving a packet knows where that packet came from. In total, our protocol uses headers with at most $O(m(\log\log n + \log m))$ bits.

At the start, S sends a packet to each node in column 0. The header of each packet contains the following information in its only stack entry: *power* $= \log_2(n-1)$, *minRow* $= 0$, *maxRow* $= m-1$, *forward* $= 1$, *loopHalf* $= 1$, and *loopCounter* $= 0$. (To be consistent with other levels of recursion, we are thinking of the path from column 0 to column $n-1$ as being the second half of a $(n-1, 0, 0, m-1)$-bounded loop.)

We shall refer to the variable $d = m - \mathit{maxRow} + \mathit{minRow}$, which is equal to the recursion depth. We describe the actions of any node (r, c) that does not appear in the first or last column of $G(m, n)$. The actions of the nodes in the first (or last) column are identical, except that they do not perform the specified forwarding of packets to the left (or right, respectively). In addition, if a node in the last column of $G(m, n)$ ever receives a packet, it forwards that packet to R.

Protocol DELIVER

On receipt of a packet at node (r, c) with (*power*, *minRow*, *maxRow*, *loopCounter*, *loopHalf*, *forward*) at the top of its stack

/* The default move is to forward a packet up, down, and in the current direction of travel. */

- If $r < \mathit{maxRow}$ and the packet was not received from node $(r+1, c)$, send the packet to node $(r+1, c)$.
- If $r > \mathit{minRow}$ and the packet was not received from node $(r-1, c)$, send the packet to node $(r-1, c)$.
- If $\mathit{power} > \#_2(c)$, then send the packet to node $(r, c + \mathit{forward})$.

/* In addition, we may choose to start a set of loops starting at the current column. This can happen only if $r > \mathit{minRow}$ or $r < \mathit{maxRow}$, either of which implies that $d < m$. */

- If $power > \#_2(c)$ and $r > minRow$, then, for $f = \pm 1$, send the packet to node $(r, c + f)$ with $(\#_2(c), minRow + 1, maxRow, 0, 0, f)$ pushed onto its stack.
- If $power > \#_2(c)$ and $r < maxRow$, then, for $f = \pm 1$, send the packet to node $(r, c + f)$ with $(\#_2(c), minRow, maxRow - 1, 0, 0, f)$ pushed onto its stack.

/* If a loop is in its first half, it can switch to the second half at any step. */

- If $loopHalf = 0$, let $minRow'$ denote the value of $minRow$ at the previous level of recursion (i.e. in the record second from the top of the stack).
  If $minRow = minRow'$
  - then send the packet to node $(r+1, c)$ with ($power$, $minRow$+1,$maxRow$+1,$loopCounter$,1,$-forward$) replacing the top record on its stack.
  - else send the packet to node $(r-1, c)$ with ($power$, $minRow$−1,$maxRow$−1,$loopCounter$,1,$-forward$) replacing the top record on its stack.

/* If a packet has returned to the column where it started its current set of loops, it has two options. */

- If $\#_2(c) \geq power$ and $loopHalf = 1$ then
  /* Option 1: start the next loop in the set. Note that if the second half of the previous loop allows the use of rows r_1 to r_2, then the previous level of the recursion allows the use of either rows r_1 to $r_2 + 1$ or rows $r_1 - 1$ to r_2. In the first case, the first half of the next loop can use either rows r_1 to r_2 or rows $r_1 + 1$ to $r_2 + 1$. In the second case, the first half of the next loop can use either rows r_1 to r_2 or rows $r_1 - 1$ to $r_2 - 1$. */
  - If $loopCounter < maxRow - minRow - 1$, then
    * For $f = \pm 1$, send the packet to node $(r, c + f)$ with ($power$, $minRow$, $maxRow$, $loopCounter$+ 1, 0, f) replacing the top record on its stack.
    * Let $minRow'$ and $maxRow'$ denote the value of $minRow$ and $maxRow$ at the previous level of recursion (i.e. in the record second from the top of the stack).
    * If $minRow = minRow'$ and $r > minRow$ then for $f = \pm 1$, send the packet to node $(r, c + f)$ with ($power$, $minRow + 1$, $maxRow + 1$, $loopCounter + 1$, 0, f) replacing the top record on its stack.
    * If $maxRow = maxRow'$ and $r < maxRow$ then for $f = \pm 1$, send the packet to node $(r, c + f)$ with ($power$, $minRow - 1$, $maxRow - 1$, $loopCounter + 1$, 0, f) replacing the top record on its stack.

  /* Option 2: stop the current set of loops and return to the previous level of the recursion. */
  - If $d > 1$, pop one record off the stack. Let $forward'$ denote the value of $forward$ at the new top level of the stack. Send the resulting packet to node $(r, c + forward')$.

End of protocol.

Lemma 3. *The header of any packet produced by the Protocol* **DELIVER** *has a length of at most* $m(\lfloor \log_2 \log_2(n-1) \rfloor + 3\lceil \log_2 m \rceil + 3) + 2$ *bits.*

Proof. It is easily verified that the maximum depth of the recursion produced by Protocol **DELIVER** is m. For each such level, the variable *power* can be represented using $\lfloor \log_2 \log_2(n-1) \rfloor + 1$ bits, the variables *maxRow*, *minRow*, and *loopCounter* can be represented using $\lceil \log_2 m \rceil$ bits, and *forward* and *loopHalf* can each be represented using a single bit. The final two bits come from the fact that each transmission informs the recipient of the direction from which the packet came. □

Lemma 4. *Protocol* **DELIVER** *transmits only a finite number of packets.*

Proof. We provide a potential function Φ for any packet in the system, such that there is a maximum value that Φ can attain and, every time a packet is forwarded, the corresponding value of Φ is increased by at least 1. (That is, each packet P has a potential exceeding the potential of the packet whose arrival caused P to be sent.) For each level of recursion i, $1 \leq i \leq m$, we define three variables: lc_i, lh_i, and $dist_i$. All of these variables are defined to be 0 if $i > d$, the current recursion depth. For $i \leq d$, lc_i and lh_i are the *loopCounter* and *loopHalf* variables, respectively, for level i in the recursion. For $i \leq d$, the variable $dist_i$ is the number of horizontal steps taken by the packet starting from the time that the *forward* variable at the i'th level of recursion was last set, counting only those steps that occurred when $d = i$. Note that a packet can only move horizontally in the direction specified by the *forward* variable, and thus all of these steps will be in the same direction. This means that $dist_i \leq n$. We also define the variable *vert* to be the number of steps taken in a vertical direction on the current column since last moving there from another column.

The potential function Φ that we define can be thought of as a $(3m+1)$-digit mixed radix number, where for $t \in \{1, \ldots, m\}$, digit $3(t-1)+1$ is lc_t, digit $3(t-1)+2$ is lh_t, and digit $3(t-1)+3$ is $dist_t$. Digit $3m+1$ is *vert*. It is easily verified that when a packet is first sent, $\Phi \geq 0$. Also, by checking each of the possible actions of a node on the receipt of a packet, we can verify that every time a packet is forwarded, Φ increases by at least 1. We also see that Φ is bounded, since $vert \leq m-1$ and, for any i, $lc_i \leq m$, $dist_i \leq n$, and $lh_i \leq 1$. Since each packet receipt causes at most a constant number of new packets to be sent out, it follows that the total number of packets sent as a result of Protocol **DELIVER** is finite. □

It follows from the decomposition of simple S–R paths given by Lemmas 1 and 2 that, if there is a simple path of live edges from S to R, then Protocol **DELIVER** finds it. We combine Lemmas 3 and 4 to get our main result.

Theorem 1. *Protocol* **DELIVER** *delivers a message from S to R with finite traffic using $O(m(\log \log n + \log m))$-bit headers .*

3 A Lower Bound

In this section, we prove that $\Omega(\log\log n)$ header bits are necessary for communicating a single message in a $3 \times n$ grid. First, we consider the graph $G(3, n)$ with $n = h!$. The proof is similar in flavour to the lower bound for communicating a single message in a complete graph [AF99].

Our proof uses the following definitions. An *S-path of extent* $j \geq 1$ is a path from $(0, c)$ to $(2, c + j - 1)$, for some column c, where $0 \leq c \leq n - j$. It consists of

- A left-to-right path of length $j - 1$ along the bottom row from $(0, c)$ to $(0, c + j - 1)$, followed by
- the vertical edge from $(0, c + j - 1)$ to $(1, c + j - 1)$, followed by
- a right-to-left path of length $j - 1$ along the middle row from $(1, c + j - 1)$ to $(1, c)$, followed by
- the vertical edge from $(1, c)$ to $(2, c)$, followed by
- a left-to-right path of length $j-1$ along the top row from $(2, c)$ to $(2, c+j-1)$.

Thus, an S-path of extent j contains $3(j - 1)$ horizontal edges and 2 vertical edges, for a total length of $3j - 1$. Similarly, a Z-*path of extent* j is a simple path of total length $3j - 1$ from $(2, c)$ to $(2, c + j - 1)$, to $(1, c + j - 1)$, to $(1, c)$, to $(0, c)$, and finally to $(0, c + j - 1)$.

Our proof focusses attention on h particular simple S–R paths, defined as follows. For $k = 1, \ldots, h$, let P_k consist of $k!$ alternating S-paths and Z-paths, each of extent $h!/k!$, concatenated using single horizontal edges. Figure 2 shows paths P_1, P_2, and P_3 for the case $h = 3$.

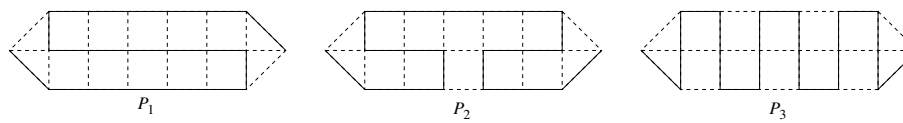


Fig. 2. Paths P_1, P_2, and P_3 for $h = 3$

For $0 \leq i < n$, let $i_1, \ldots, i_h$ be such that $i = \sum_{k=1}^{h} i_k n/k!$ where $0 \leq i_k < k$. In other words, $(i_1, \cdots, i_h)$ is the mixed radix representation of i, where the k'th most significant digit is in base k. Note that i_1 always has value 0. For example, if $n = 24 = 4!$ and $i = 20$, then $i_1 = 0$, $i_2 = 1$, $i_3 = 2$, and $i_4 = 0$.

Proposition 2. *Let* $0 \leq i < j < n$. *Node* $(1, j)$ *appears before node* $(1, i)$ *in path* P_k *if and only if* $i_d = j_d$ *for* $d = 1, \ldots, k$.

Proof. In every S-path or Z-path, the nodes in row 1 appear in order from largest numbered column to smallest numbered column. Since path P_k is the concatenation of S-paths and Z-paths, node $(1, j)$ appears before node $(1, i)$ if

and only if columns i and j are in the same S-path or Z-path. Since each S-path and Z-path comprising P_k has extent $n/k!$, it follows that i and j are in the same S-path or Z-path if and only if $\lfloor i/(n/k!) \rfloor = \lfloor j/(n/k!) \rfloor$, which is true if and only if $i_d = j_d$ for $d = 1, \ldots, k$. □

Consider any protocol for $G(3, h!)$ that delivers a message from S to R with finite traffic. Since node $(1, c)$ is on path P_k, it receives at least one packet when only the links on the simple S–R path P_k are alive. Let $H_k(c)$ denote the header of the last packet received by node $(1, c)$ in this situation that causes a packet to be received by R.

Lemma 5. *Consider any protocol for $G(3, h!)$ that delivers a message from S to R with finite traffic. Then, for all path indices $1 \leq j < k \leq h$ and all columns $0 \leq c < c' < h!$ such that $(c_1, c_2, \ldots, c_j) = (c'_1, c'_2, \ldots, c'_j)$ and $(c_1, c_2, \ldots, c_k) \neq (c'_1, c'_2, \ldots, c'_k)$, either $H_j(c) \neq H_k(c)$ or $H_j(c') \neq H_k(c')$.*

Proof. To obtain a contradiction, suppose that $H_j(c) = H_k(c)$ and $H_j(c') = H_k(c')$, for some path indices $1 \leq j < k \leq h$ and some columns $0 \leq c < c' < h!$ such that $(c_1, c_2, \ldots, c_j) = (c'_1, c'_2, \ldots, c'_j)$ and $(c_1, c_2, \ldots, c_k) \neq (c'_1, c'_2, \ldots, c'_k)$. Then, by Proposition 2, node $(1, c')$ appears before node $(1, c)$ in path P_j but after node $(1, c)$ in path P_k.

Consider the situation when the links on both paths P_j and P_k are alive. The protocol forwards a packet along path P_k until a packet with header $H_k(c')$ reaches node $(1, c')$. This causes a packet to be received by R. Since $H_k(c') = H_j(c')$ and node $(1, c')$ occurs before node $(1, c)$ on path P_j, it also causes a packet with header $H_j(c)$ to be received at node $(1, c)$. Likewise, since $H_j(c) = H_k(c)$ and node $(1, c)$ occurs before node $(1, c')$ on path P_k, this causes a packet with header $H_k(c')$ to be received at node $(1, c')$, and we have an infinite loop. Each time such a packet goes through the loop, it produces a new packet that is sent to the destination R. This contradicts the finite-traffic assumption. □

Lemma 6. *Consider any protocol for $G(3, h!)$ that delivers a message from S to R with finite traffic. Then, for $1 \leq k \leq h$, there exist nonnegative digits $i_1 < 1, i_2 < 2, \ldots, i_k < k$ such that the k headers $H_1(c), \ldots, H_k(c)$ are distinct for each column c with $(c_1, c_2, \ldots, c_k) = (i_1, i_2, \ldots, i_k)$.*

Proof. To obtain a contradiction, suppose the lemma is false. Consider the smallest value of $k \leq h$ for which the lemma is false. Since there are no repetitions in a sequence of length one, $k > 1$. Let $i_1 < 1, i_2 < 2, \ldots, i_{k-1} < k - 1$ be such that the $k - 1$ headers $H_1(c), \ldots, H_{k-1}(c)$ are distinct for each column c with $(c_1, c_2, \ldots, c_{k-1}) = (i_1, i_2, \ldots, i_{k-1})$. Then, for each digit $i_k \in \{0, \ldots, k - 1\}$, there exists a path index $j \in \{1, \ldots, k - 1\}$ and a column c such that $(c_1, c_2, \ldots, c_{k-1}, c_k) = (i_1, i_2, \ldots, i_{k-1}, i_k)$ and $H_k(c) = H_j(c)$.

Since there are k choices for i_k and only $k - 1$ choices for j, the pigeonhole principle implies that there exist distinct $i_k, i'_k \in \{0, \ldots, k-1\}$ which give rise to the same value of j and there exist columns c and c' such that $(c_1, c_2, \ldots, c_{k-1}) = (c'_1, c'_2, \ldots, c'_{k-1})$, $c_k = i_k \neq i'_k = c'_k$, $H_k(c) = H_j(c)$, and $H_k(c') = H_j(c')$. But this contradicts Lemma 5. □

Theorem 2. *Any protocol for $G(3,n)$ that delivers a message from S to R with finite traffic uses headers of length at least* $\log_2 \log_2 n - O(\log \log \log n)$.

Proof. Let h be the largest integer such that $n \geq h!$. Then $n < (h+1)! < (h+1)^h$, so $h \log_2(h+1) > \log_2 n$ and $h \in \Omega(\log n / \log \log n)$.

Consider any protocol for $G(3,n)$ that uses headers of length L. Since $G(3,h!)$ is a minor of $G(3,n)$, it follows from Proposition 1 that there is a protocol for $G(3,h!)$ using headers of length L. Hence, by Lemma 6, $L \geq \log_2 h = \log_2 \log_2 n - O(\log \log \log n)$. □

References

AF99. M. Adler and F.E. Fich, *The Complexity of End-to-End Communication in Memoryless Networks*, Proceedings of the 8th Annual ACM Symposium on Principles of Distributed Computing, (1999), 239–248.

AAF+94. Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D. Wang, and L. Zuck, *Reliable Communication Over Unreliable Channels*, Journal of the ACM, **41(6)**, (1994), 1267–1297.

AAG+97. Y. Afek, B. Awerbuch, E. Gafni, Y. Mansour, A. Rosén, and N. Shavit, *Slide–The Key to Polynomial End-to-End Communication*, Journal of Algorithms, **22(1)**, (1997), 158–186.

AG88. Y. Afek and E. Gafni, *End-to End Communication in Unreliable Networks*, Proceedings of the 7th Annual ACM Symposium on Principles of Distributed Computing, (1988), 131–148.

AMS89. B. Awerbuch, Y. Mansour, and N. Shavit, *Polynomial End to End Communication*, Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, (1989), 358–363.

APV96. B. Awerbuch, B. Patt-Shamir, and G. Varghese, *Self-stabilizing End-to-End Communication*, Journal of High Speed Networks, **5(4)**, (1996), 365–381.

DW97. S. Dolev and J. Welch, *Crash Relient Communication in Dynamic Networks*, IEEE Transactions of Computers, **46**, (1997), 14–26.

F98. F.E. Fich, *End-to-end Communication*, Proceedings of the 2nd International Conference on Principles of Distributed Systems, (1998), 37–43.

K30. K. Kuratowski, *Sur le Problème des Courbes Gauches en Topologie*, Fund. Math., **15**, (1930) 271–283.

KOR95. E. Kushilevitz, R. Ostrovsky, and A. Rosén, *Log-Space Polynomial End-to-End Communication*, Proceedings of the 28th ACM Symposium on the Theory of Computing, (1995), 559–568.

LLT98. R. Ladner, A. LaMarca, and E. Tempero, *Counting Protocols for Reliable End-to-End Transmission*, Journal of Computer and Systems Sciences, **56(1)**, (1998), 96–111.

P81. J. Postel, *Internet Protocol*, Network Working Group Request for Comments 791, September 1981.

Wavelength Assignment Problem on All-Optical Networks with k Fibres per Link

Luciano Margara[1] and Janos Simon[2]

[1] Computer Science Department, University of Bologna. margara@cs.unibo.it
[2] Computer Science Department, University of Chicago. simon@cs.uchicago.edu

Abstract. Given a (possibly directed) network, the *wavelength assignment problem* is to minimize the number of wavelengths that must be assigned to *communication paths* so that paths sharing an edge are assigned different wavelengths. Our generalization to multigraphs with k parallel edges for each link (k fibres per link, with switches at nodes) may be of practical interest. While the wavelength assignment problem is NP-hard, even for a single fibre, and even in the case of simple network topologies such as rings and trees, the new model suggests many nice combinatorial problems, some of which we solve. For example, we show that for many network topologies, such as rings, stars, and specific trees, the number of wavelengths needed in the k-fibre model is *less* than $1/k$ fraction of the number required for a single fibre. We also study the existence and behavior of a *gap* between the minimum number of wavelengths and the natural lower bound of *network congestion*, the maximum number of communication paths sharing an edge. For optical stars (any size) while there is a $3/2$ gap in the single fibre model, we show that with 2 fibres the gap is 0, and present a polynomial time algorithm that finds an optimal assignment. In contrast, we show that there is no fixed constant k such that for every ring and every set of communication paths the gap can be eliminated. A similar statement holds for trees. However, for rings, the gap can be made arbitrarily small, given enough fibres. The gap can even be eliminated, if the length of communication paths is bounded by a constant. We show the existence of *anomalies*: increasing the number of fibres may increase the gap.

1 Introduction

We study a collection of interesting combinatorial questions, motivated by optimization problems in the context of *optical interconnection networks*. For the purposes of this paper, an all-optical network consists of *routing nodes* interconnected by point-to-point fibre-optic *links*, which can support a certain number of wavelengths. Links are bidirectional. Each message travels through the network on a specific wavelength that, in this model, *cannot be changed* during the transmission. Two variants of this model of optical networks have been studied intensively (see [2] for an up-to-date survey): the *directed* model, in which two messages traveling on the same fibre-optic link in the same direction must have different wavelengths, and the *undirected* model, in which two messages passing

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 768–779, 2000.
© Springer-Verlag Berlin Heidelberg 2000

through the same fibre-optic link must have different wavelengths no matter in which direction they are traveling [1] In what follows, a message will be called a *request* and a set of requests will be called an *instance*. Given a network G and an instance I on G, the *wavelength-routing problem* consists of finding a *routing scheme* R for I and an assignment of a wavelength to each request of I, such that no two paths of R *sharing an edge* have the same wavelength, and such that the total number of wavelengths is minimized. In this paper we do not address the problem of obtaining a good routing scheme for a set of requests on a given network. We will assume that the routing scheme is given as part of the input, or that it is uniquely determined by the topology of the network we are considering (as in the case of optical trees). We will focus on the problem of assigning wavelengths to a given set of communication paths. The problem of finding an optimal wavelength assignment is NP-hard even if we restrict our attention to very simple network families such as *rings* or *trees* [4,5,6]. In some cases, there exist polynomial time greedy algorithms (see for example [16,9]) which provide approximate solutions for this problem in terms of the *network congestion* (the maximum number of connection paths which share a fibre-optic link) which, in turn, is a lower bound on the optimal number of wavelengths.

We define and analyze a new optical network model: Each point-to-point fibre-optic link consists of k distinct optical fibres (the same k for each link). This assumption is very natural [2], and suggests many interesting algorithmic and combinatorial questions. In this new model, each time we send a message through a link we need to specify which fibre we want to use. Two paths sharing a link can be given the same wavelength if they pass through distinct fibres.

We ask the following *basic question*: can we reduce the number of wavelengths by a factor strictly larger than k using k fibres per link ? We prove that for a number of network topologies of practical interest, this question has an affirmative answer. We also identify many challenging and (in our view) interesting problems, and provide solutions to some of them.
The main results of this paper are:

– We show (Thm. 4) that for any $k, m \geq 1$ there exists a network G, an instance I on G, and a routing R for I, such that the minimal number of wavelengths needed using k fibres per link is at least m, while the number of wavelengths needed using $k+1$ fibres per link is 1. Note that this gives an affirmative answer our *basic question* for instance I.

– For optical *star* networks we are able to show significant improvement by using multiple fibres. In the undirected single fibre model every instance can be routed using a number of wavelengths equal to 3/2 times the congestion of the network [15] and this is the best ratio achievable. In contrast using 2

[1] Brief justification for the models: physical links (fibres) are undirected. Current repeaters and routers aren't.

[2] We do not discuss practicality: it is easy to justify wiring that uses multiple fibres, and it is possible to build appropriate switches. Whether such switches can be made economical is not clear. This may also depend on the benefits of multiple fibres. We hope that papers like ours will eventually determine the amount of these benefits.

fibres per link it is possible to route any set of requests on an undirected star network optimally (i.e. using a number of wavelengths equal to the congestion of the network, where the congestion of the network using k fibres is defined as the largest number of paths through any fibre–equivalently, this is the value of the congestion of the single fibre version of the same network, divided by k). Moreover, we give a polynomial time algorithm (Thm. 5) that assigns paths to fibres.

– In the case of optical *tree* networks we prove that there is no single constant k so that for every tree all instances can be routed using a number of wavelengths equal to the congestion of the busiest link. This is true both for undirected (Thm. 9) and directed (Thm. 10) networks. The theorem holds even if we restrict the underlying graph to be the family of undirected trees of height 2. Note that this does not mean that it is impossible to eliminate the gap for a fixed graph. In fact, we prove that for binary trees of height 2, 4 fibres are enough for closing the gap between number of wavelengths and network congestion (Thm. 7).

– For *ring* networks we give a polynomial time algorithm (Thm. 12) which takes as input an optical ring G with n nodes (either directed or undirected), an instance I on G, and a routing scheme R for I and computes a wavelength assignment for R whose cardinality is at most $1 + 1/k$ times larger than the congestion caused by R on G, where k is the number of fibres per link. Note that using one fibre per link the best ratio achievable between number of wavelengths and network congestion is 2 [16]. We also prove (Thm. 13) that for every $k \geq 1$ there exist an optical ring G with n nodes and k fibres per link, an instance I on G, and a routing scheme R for I such that the cardinality of any wavelength assignment for R is $2k/(2k-1) = 1 + 1/(2k-1)$ times larger than the network congestion caused by R on G. Finally, we show (Thm. 14) that if all the requests have a length uniformly bounded by an arbitrarily chosen constant c, then there exists a value k_c (which depends only on c) such that using k_c fibres per link it is possible to close the gap between the number of wavelengths used and the network congestion for ring of any size.

– We show that, perhaps surprisingly, adding fibres may *increase* the gap between load and number of wavelengths (Thm. 7).

The following question remains open: Is it true that given a fixed network G there exists a number $k \geq 1$ such that using k fibres per link it is possible to route all the possible instances on G using a number of wavelengths equal to the congestion of the network ?
Due to limited space some of the proofs are omitted, or only sketched.

2 Basic Definitions

The standard optical model. We represent an *all-optical network* as a graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ represents the set of nodes of the network and $E \subseteq \{(i, j) \mid i, j \in V\}$ represents the set of node-to-node connections available in the network. A *request* is an ordered pair of vertices (s, d), $s, d \in V$, corresponding

to a message to be sent from node s to node d. An *instance* I is a collection of requests. Note that a given request can appear more than once in the same instance. Let $I = \{(s_1, d_1), \ldots, (s_m, d_m)\}$ be an instance on G. A *routing scheme* $R = \{p_1, \ldots, p_m\}$ for I is a set of simple directed paths on G. Each path p_i is a sequence $(v_1, \ldots, v_k)$ of distinct vertices of V such that $v_1 = s_i$ and $v_k = d_i$. We say that a path $p = (v_1, \ldots, v_k)$ contains the edge (i, j) if there exists h, $1 \leq h \leq k-1$, such that $v_h = i$ and $v_{h+1} = j$. A legal *wavelength assignment* W of cardinality m for a routing scheme R is a map from R to $[1, \ldots, m]$ such that if two elements $p, q \in R$ *share an edge* then $W(p) \neq W(q)$, i.e., they are given distinct wavelengths. This defines two variant models, depending on the interpretation of "...sharing an edge...":

– **Directed model.** Two paths p and q share the edge (i, j) if both p and q contain the edge (i, j).

– **Unirected model.** Two paths p and q share the edge (i, j) if both p and q contain at least one edge in the set $\{(i, j), (j, i)\}$.

The *wavelength-routing problem* can be formulated as follows: Given a graph G and an instance I on G, find a routing scheme R for I and a legal wavelength assignment W for R such that the cardinality of W is the minimum among all possible routing schemes R for I and legal wavelength assignments W for R.

The new optical model. A *legal k-wavelength assignment* W of cardinality m for a routing scheme R is a map from R to $[1, \ldots, m]$ such that if $k+1$ elements $p_1, \ldots, p_{k+1} \in R$ *share an edge* then there exist $1 \leq i, j \leq k+1$ such that $W(p_i) \neq W(p_j)$.

Our definition is equivalent to considering a multigraph obtained from G by replacing every edge by a set of k parallel edges. Note that we consider both directed and undirected legal k-wavelength assignments. In the directed model k paths $p_1, \ldots, p_k \in R$ share the edge (i, j) iff every p_i, $1 \leq i \leq k$, contains the edge (i, j), while in the undirected model k paths $p_1, \ldots, p_k \in R$ share the edge (i, j) if every p_i, $1 \leq i \leq k$, contains at least one edge in the set $\{(i, j), (j, i)\}$.

Number of wavelengths and network congestion. Let I be an instance on a graph $G = (V, E)$. Let $R = \{p_1, \ldots, p_m\}$ be a routing scheme for I. We define the *conflict graph* $G_c = (V_c, E_c)$ for R, and G as having vertices $V_c = R$ and edges $E_c = \{(p_i, p_j) \mid p_i \text{ and } p_j \text{ share an edge of } G\}$. We denote by $W(R, G, k)$ the cardinality of the best possible legal k-wavelength assignment for R. It is easy to verify that $W(R, G, 1)$ is equal to the chromatic number of G_c. Let $L(R, G, \alpha)$, the *load of* α be the maximum number of paths of R sharing the edge α. Let $L(R, G)$ be the maximum of $L(R, G, \alpha)$ over all the edges α of G. It is easy to verify that $W(R, G, k) \geq \lceil \frac{1}{k} L(R, G) \rceil$. In the 1-fibre case ($k$=1) $L(R, G)$ is called the *congestion* of the network. Similarly, we will call the quantity $\lceil \frac{1}{k} L(R, G) \rceil$ the k-congestion (or, when k is clear from the context) simply, *congestion*. Fix a graph G. Let $T = \{I_i\}_{i \in N}$ be a sequence of instances on G, and let $S = \{R_i\}_{i \in N}$ be a sequence of routing schemes. We say that S produces a k-gap $r \geq 1$ on G if and only if

$$\text{for every } i \geq 1: \qquad \frac{W(R_i, G, k)}{\lceil \frac{1}{k} L(R_i, G) \rceil} \geq r.$$

We denote by $Gap(S, G, k)$ the maximum k-gap that S produces on G. We define the k-gap of G, denoted by $Gap(G, k)$, as the supremum of $Gap(S, G, k)$ taken over all possible sequences S. Again, we will omit k when its value is clear from the context. We define $N(G)$ as the minimum k such that $Gap(G, k) = 1$, if such a k exists.

Acyclic networks. Acyclic networks are modeled by acyclic graphs. In acyclic graphs an instance I uniquely determines its routing scheme, so we omit R: I will denote both a set of requests and the associated set of paths.

3 More Fibres Can Help

In this section we prove that there exist a network G, an instance I on G, and a routing scheme R for I such that the ratio between the number of wavelengths needed for R using k fibres and the number of wavelengths needed for R using $k + 1$ fibres can be made arbitrarily large. We start with two observations.

Observation 1 *Let I be an instance on some graph $G = (V, E)$, and let R be a routing scheme for I. If the conflict graph G_c associated to R and G contains no triangles (cliques with 3 nodes) then $W(R, G, 2) = 1$.*

Observation 2 *Let G_c be an arbitrary graph. Then we can construct, in time polynomial in the size of G_c a graph G, an instance I on G, and a routing scheme R for I on G, such that G_c is the conflict graph of (R, G).*

We use these observations to prove the following result.

Theorem 3. *Given any $m \geq 1$ it is possible to construct a graph $G = (V, E)$, an instance I on G, and a routing scheme R for I such that: $W(R, G, 1) \geq m$ and $W(R, G, 2) = 1$.*

Sketch of proof. Let G_c be any graph with chromatic number at least m and with maximum clique at most 2 (for the construction of such a graph see for example [11]) From G_c, we construct a graph G, an instance I on G, and a routing scheme R for I on G such that G_c is the conflict graph of (R, G). Then we conclude that $W(R, G, 1) \geq m$ and $W(R, G, 2) = 1$. □

We generalize Thm. 3 as follows.

Theorem 4. *Given any $m \geq 1$ and $k \geq 2$ it is possible to construct a graph $G = (V, E)$, an instance I on G, and a routing scheme R for I such that $W(R, G, k-1) \geq m$ and $W(R, G, k) = 1$.*

Sketch of proof. It is possible to construct a network G an instance I on G, and a routing scheme R for I such that: for every subset $S \subset R$ of $k + 1$ paths there is an edge e in G such that all the paths of S share the edge e, and $L(R, G) = k + 1$. As a consequence we have that $W(R, G, k) \geq |R|/k$, while $W(R, G, k + 1) = 1$. □

4 Star Networks

An n-star is a graph $G = (V, E)$ such that $V = \{c, x_1, \ldots, x_n\}$ and $E = \{(c, x_i),\ i = 1, \ldots, n\}$. The node c is the *center* of the star, while the nodes x_i are the *leaves* of the star. In the case of star networks using the single fibre directed model it is possible to efficiently (in polynomial time) route all instances with a number of wavelengths equal to the network congestion (this problem is equivalent to computing the chromatic index of bipartite graphs [18]). This is no longer true in the undirected model. The best ratio achievable in the undirected model between number of wavelengths and network congestion is 3/2 [15]. Also, computing the optimal wavelength assignment in this model is an NP-hard problem (it is equivalent to the edge-coloring problem of multigraphs, which is an NP-hard problem [8]). In the next theorem we show a rather surprising result: using 2 fibres it is always (independently of the size of the star) possible to find a wavelength assignment whose cardinality is equal to the network congestion. Moreover, this can be done in polynomial time.

Theorem 5. *Let G be any n-star. Then, in the undirected model, $N(G) = 2$.*

Proof. Let G be an n-star with vertex set $V = \{c, x_1, \ldots, x_n\}$. Let I be any set of paths on G. We have $L(I, G) = \max\{deg(x_i) \mid i = 1, \ldots, n\}$, where $deg(x_i)$ is the number of paths touching the node x_i. Without loss of generality we assume $L(I, G)$ even. We prove that there exists a legal 2-wavelength assignment for G of cardinality $L(I, G)/2$. We first add to I as many new paths as we can without increasing $L(I, G)$. At the end of this procedure each node has degree $L(I, G)$ except for at most one node. For assume this is not the case. Then there are two distinct nodes x_i and x_j of less than maximum degree. Adding the path (x_i, x_j) to I does not increase $L(I, G)$, contradicting the maximality of I. Assume that x_i is the only node with degree $d < L(I, G)$. Since $n - 1$ nodes have degree $L(I, G)$ and x_i has degree d and since each path has 2 endpoints we know that $(n-1)L(I, G) + d$ is even. Since $L(I, G)$ is even we conclude that also d is even. We now add two new nodes x_{n+1} and x_{n+2} to G. Then we add to I:
– $(L(I, G) - d)/2$ paths from x_i to x_{n+1},
– $(L(I, G) - d)/2$ paths from x_i to x_{n+2}, and
– $(L(I, G) + d)/2$ paths from x_{n+1} to x_{n+2}.
Now, all the nodes of G have even degree $L(I, G)$. Consider a new graph $G' = (V', E')$ where

$$V' = V \setminus \{c\} \quad \text{and} \quad E' = \{(x_i, x_j) \mid \text{ the path from } x_i \text{ to } x_j \text{ belongs to } I\}.$$

G' is $L(I, G)$-regular (each node has degree $L(I, G)$) with $L(I, G)$ even, so E' can be partitioned [13] into $L(I, G)/2$ subsets $E'_1, \ldots, E'_{L(I,G)/2}$ such that each graph $G' = (V', E'_i)$ is 2-regular. Let $I_i \subseteq I$ be the set of paths corresponding to E'_i. It is easy to verify that $L(I_i, G) = 2$ and therefore $W(I_i, G, 2) = 1$. This implies that there exists a legal 2-wavelength assignment W for (I, G) with $|W| = L(I, G)/2$ as claimed.

Since there is a polynomial time algorithm to partition a graph into 2-factors (2-regular spanning subgraphs) [1], and the other procedures used in the constructive proof above are clearly in polynomial time we have:

Corollary 6. *Let G be any n-star and I be any instance on G. An optimal wavelength assignment for (I, G) in the 2-fibres undirected model can be computed in polynomial time.*

5 Tree Networks

We first consider the well studied H-graph network [10], the complete binary tree of height 2 in the undirected model. We show that the H-graph is the simplest network that needs more than 2 fibres per link to close the gap between number of wavelengths and network congestion: 4 fibres are necessary and sufficient. We also show that the H-graph is a simple example of a network that exhibits a *monotonicity anomaly*: using more fibres may *increase* the gap.

Theorem 7. *Let G be an H-graph. In the undirected model we have: $N(G) = 4$, and $Gap(G, 5) > 1$.*

The proof, a small explicit instance, will be presented in the full journal version.

If only leaf-to-leaf communication paths are allowed then it is possible to prove that $N(G) = 2$ in the undirected model and $N(G) = 1$ in the directed model. We now prove that, in a directed H-graph, 2 fibres per link are not enough for closing the gap between number of wavelengths and network congestion. The problem of determining the minimum number of fibres necessary for closing the above mentioned gap in a directed H-graph is still open.

Theorem 8. *Let G be a directed H-graph. Then $N(G) \geq 3$.*

Proof. We construct a sequence S of instances such that $Gap(S, G, 2) \geq \frac{9}{8}$. Let x be the root of G. Let y_1 and y_2 be the left and the right child of x. Let z_1, z_2, z_3, and z_4 be the leave s of G listed from left to right. Let I be defined as follows: 3 copies of path (z_1, z_2), 2 copies of path (z_2, z_1), 3 copies of path (z_3, z_4), 2 copies of path (z_4, z_3), 2 copies of path (z_2, z_3), 1 copy of path (z_3, z_2), 1 copy of path (z_1, y_2), 1 copy of path (y_1, z_4), 1 copy of path (y_2, z_1), 1 copy of path (z_4, y_1), and 1 copy of path (z_4, z_1).

We have $|I| = 18$ and $L(I, G) = 4$. Let I_j be a set of paths consisting of j copies of I. Let $S = \{I_j\}_{j \in N}$. It is easy to verify that $|I_j| = \frac{9}{2} L(I_j, G)$. Since the largest subset Q of I_j such that $W(Q, G, 2) = 1$ has cardinality 8 (this can be proven by exhaustive analysis), we have

$$W(I_j, G, 2) \geq \frac{|I_j|}{8} = \frac{9}{16} L(I_j, G) \quad \text{or, equivalently,} \quad \frac{W(I_j, G, 2)}{\frac{1}{2} L(I_j, G)} \geq \frac{9}{8}.$$

By definition of Gap we conclude that $Gap(S, G, 2) \geq \frac{9}{8}$, so $Gap(G, 2) \geq \frac{9}{8}$.

For general optical trees of height 2 we have the following result.

Theorem 9. *For every $k \geq 1$ there exists an undirected optical tree T of height 2 such that $Gap(T,k) \geq 1 + \frac{1}{2k^2}$.*

Proof. Let $k \geq 1$ be an integer. We define an undirected optical tree T of height 2 as follows. X is the root of T with left child L and right child R; L has $2k$ children $l_0, \ldots, l_{2k-1}$ and R has $2k$ children $r_0, \ldots, r_{2k-1}$. We define on T a set P of undirected paths as follows. P consists of $2k-1$ copies of $left-short-paths$, $2k-1$ copies of $right-short-paths$, and 1 copy of $long-paths$, where

$$\begin{aligned} left-short-paths &= \{(l_0,l_1),(l_2,l_3),\ldots,(l_{2k-2},l_{2k-1})\} \\ right-short-paths &= \{(r_0,r_1),(r_2,r_3),\ldots,(r_{2k-2},r_{2k-1})\} \\ long-paths &= \{(l_i,r_i) \mid i=2h,\ h=0,1,\ldots,k-1\} \cup \\ &\qquad \left\{(l_i, r_{(i+2) \bmod 2k}) \mid i=2h+1,\ h=0,1,\ldots,k-1\right\} \end{aligned}$$

The cardinality of P is $4k^2$ and $L(P,T) = 2k$. Let I_j be the set of paths on T obtained by taking j copies of P. Trivially, the cardinality of I_j is $4jk^2$ and $L(I_j,T) = 2jk$. Let $S = \{I_i\}_{i \in N}$. Let P' be any subset of I_j such that $W(P',T,k) = 1$. We claim that the cardinality of P' is at most $2k^2$ and that if P' contains at least one long path, then the cardinality of P' is at most $2k^2-1$. To prove our claim we proceed as follows. Let P' be the union of k copies of the paths in the set $left-short-paths$ and k copies of the paths in the set $right-short-paths$. It is not difficult to prove that $W(P',T,k) = 1$ and that the cardinality of P' is $2k^2$. If we insert a long path in P', in order to maintain $W(P',T,k) = 1$, we are forced to remove 2 short paths from P' decreasing its cardinality by one. We can insert in P' at most $k-1$ other long paths without violating the property $W(P',T,k) = 1$. Each insertion of a long path in P' forces us to remove at least one short path. This completes the proof of our claim.

I_j contains $2jk$ long paths. Since $W(P',T,k) = 1$, P' contains at most k long paths. This means that for assigning wavelengths to all long paths we need at least $2j$ distinct new wavelengths. We call them long wavelengths. Each long wavelength can be given to at most $2k^2-1$ paths in I_j. This means that to complete the wavelength assignment of I_j we still have to assign wavelengths to at least $4jk^2 - 2j(2k^2-1)$ paths and then we need at least $\frac{4jk^2-2j(2k^2-1)}{2k^2} = \frac{j}{k^2}$ new wavelengths. So $W(I_j,T,k) \geq 2j + \frac{j}{k^2}$. Since $\frac{1}{k}L(I_j,T) = 2j$, we conclude that $Gap(S,T,k) \geq 1 + \frac{1}{2k^2}$ as claimed.

A similar result can be proven in the directed model.

Theorem 10. *For every $k \geq 1$ there exists a directed optical tree T of height 2 such that $Gap(T,k) \geq 1 + \frac{1}{4k^2}$.*

We omit the proof of this theorem since it is similar to the proof of Theorem 9.

6 Ring Networks

An n-ring is a graph $G = (V,E)$ with
$V = \{x_0, \ldots, x_{n-1}\}$ and $E = \{(x_i, x_{i+1}),\ i = 0, \ldots, n-2\} \cup \{(x_{n-1}, x_0)\}$.

For ring networks, if we are not interested in routing schemes, there is no difference between the directed and the undirected model. In fact, once we are given a routing scheme R for an instance I in a directed optical ring, the set of paths of R can be partitioned into two disjoint sets of paths, C, paths routed in the clockwise direction and CC, routed in the opposite direction. Since there are no conflicts among paths belonging to C and CC (they use different directions on every edge), the original problem is equivalent to the two undirected wavelength assignment problems given by the set of requests in C and CC. For this reason, we will consider only the problem of assigning wavelengths to a set I of undirected paths on a ring.
In the proof of next theorem we give an algorithm which takes as input an optical ring G of any size and an instance I on G and produces a 2-legal wavelength assignment whose cardinality is at most 3/2 larger than the network congestion caused by I (plus 3).

Theorem 11. *There exists a polynomial time algorithm which, given any set I of paths on any n-ring G, produces a legal 2-wavelength assignment W such that* $|W| \leq \frac{3}{2}\frac{L(I,G)}{2} + 3$.

Sketch of proof. Let $G=(V,E)$ be an n-ring, with vertex set $V=\{x_0,\ldots,x_{n-1}\}$ and edge set $E=\{(x_i,x_{i+1}),\ i=0,\ldots,n-2\}\cup\{(x_{n-1},x_0)\}$. Let I be any set of paths on G. Without loss of generality, we assume that:
– each path of I starting at x_i and ending at x_j (denoted by (x_i,x_j), passes through nodes $x_{i+1},x_{i+2},\ldots,x_{j-1}$, where sums are taken modulo n,
– each edge $e\in E$ has full load $L=L(I,G)$, i.e., exactly $L(I,G)$ paths pass through e, and
– there are no paths starting from or ending at x_0.

We say that a path (x_i,x_j) is a *regular* path if $i<j$ and is a *crossing* path if $i>j$. Let $cp_1,\ldots,cp_L$ be the crossing paths for (I,G). Our algorithm computes a 2-legal wavelength assignment W as follows. We first assume that every crossing path $cp_i=(s_i,d_i)$ can be given two different wavelengths at the same time. The first wavelength is associated to the segment $s_i,\ldots,x_0$ and the second wavelength is associated to the segment $x_0,\ldots,d_i$. Taking advantage of this (illegal) assumption, it is possible to find a legal 1-wavelength assignment W' such that $|W'|=L$ (this problem is equivalent to the wavelength assignment problem on the line). W' applied to a regular path returns a single wavelength, while W' applied to a crossing path returns a pair of wavelengths associated to the first and to the second segment of the path. We say that a subset $S=\{cp_{i_1},\ldots,cp_{i_h}\}$ of crossing paths is *cyclic* according to W' iff :
– for every $j=1,\ldots,h-1$, the wavelength associated to the second segment of cp_{i_j} is equal to the wavelength associated to the first segment of $cp_{i_{j+1}}$ and
– the wavelength associated to the second segment of cp_{i_h} is equal to the wavelength associated to the first segment of cp_{i_1}.

We now partition the set of crossing paths into cyclic subsets $S_1,\ldots,S_m$. Note that this decomposition is unique up to a relabeling of the cyclic subsets. Consider now a generic cyclic set S_i having cardinality $4h$ for some $h\geq 1$. Let W_i'

be the set consisting of the $4h$ distinct wavelengths associated to the crossing paths belonging to S_i. Let $I_i \subseteq I$ be the set of paths having a wavelength belonging to W'_i. It is easy to verify that $L(I_i, G) = 4h$ and that $L(I \setminus I_i, G) = L - 4h$. We claim that there exists a wavelength assignment W for the paths in I_i with cardinality $3h$. To prove this fact we proceed as follows.
Step 1. To each crossing path $cp_{i_{2j+1}},\ j = 0, \ldots, 2h-1$ of S_i we assign a new wavelength w. We assign w also to each path in I_i whose wavelength according to W' is one of the two wavelengths associated to the two segments of $cp_{i_{2j+1}}$. Globally, in Step 1, we use $2h$ new wavelengths.
Step 2. To each pair of crossing paths $cp_{i_{4j+2}}$ and $cp_{i_{4j+4}},\ j = 0, \ldots, h-1$ of S_i we assign a new wavelength. Globally, in Step 2, we use h new wavelengths.
The wavelength assignment W defined in Step 1 and 2 has the following properties: $|W| = 3h$, W is a 2-legal assignment for (I_i, G), $L(I \setminus I_i, G) = L - 4h$, and $|W| = (3/2)(L(I_i, G)/2)$.

Assume for a moment that all S_is have cardinalities multiple of 4. In this easy case we just have to repeat Step 1 and 2 until all cyclic sets have been considered. Unfortunately, in general not all S_is have cardinality $4h$ for some $h \geq 1$. If the cardinality of $|S_i| = 4h + d,\ d = 1, 2, 3$, we need to construct a wavelength assignment W in a slightly more complicated way. We now describe the basic idea for dealing with the case $|S_i| = 4h + 1$. The other 2 cases can be treated analogously and the complete proof will be given in the full paper.
If $|S_i| = 4h + 1$, we distinguish two cases. If there exists another S_j such that $|S_j| = 4h+1$, we construct W for $S_i \cup S_j$ using $6h$ wavelengths using a technique which is similar to that used in Step 1 and 2. If S_i is the unique cyclic set with cardinality $4h+1$ then we construct W for S_i alone using $3h+1$ wavelengths. This is one of the three cases in which we are forced to use one additional wavelength. This completes the sketch of the proof. □

The result given in Thm. 11 for the 2 fibres model can be generalized to the k fibres model using a similar proof technique.

Theorem 12. *There exists a polynomial time algorithm which, given any instance I on any n-ring G, produces a legal k-wavelength assignment W such that $|W| \leq \left(1 + \frac{1}{k}\right) \frac{L(I,G)}{k} + c_k$, where c_k depends on k but is independent of I and n (and then $L(I, G)$).*

In the next theorem we prove that if we consider optical rings of any size with k fibres per link, the ratio between wavelengths and network congestion can be made arbitrarily close to $\frac{2k}{2k-1}$. As a consequence, we have that, no matter how many fibres we use, it is impossible to close the gap between wavelengths and network congestion (as we did in the case of optical stars) for the entire family of optical rings at the same time.

Theorem 13. *Let A_k be any algorithm such that for every n-ring G and every instance I on G finds a legal k-wavelength assignment W for (I, G) such that $|W| \leq \alpha \frac{L(I,G)}{k}$. Then $\alpha \geq \frac{2k}{2k-1}$.*

Sketch of proof. Let $G_n = (V, E)$ be an n-ring with n even, where $V = \{x_0, \ldots, x_{n-1}\}$ and $E = \{(x_i, x_{i+1}),\ i = 0, \ldots, n-2\} \cup \{(x_{n-1}, x_0)\}$. Let $I_n = P_1 \cup P_2$ where

$$P_1 = \left\{(x_i, x_{i+n/2}) \mid 0 \le i \le n/2 - 1\right\}, P_2 = \left\{(x_{i+n/2}, x_{i+1}) \mid 0 \le i \le n/2 - 2\right\}.$$

It is easy to see that $L(I_n, G_n) = n/2$ and that $|I_n| = n-1$, while it takes a little effort to prove that the largest subset Q of I_n such that $W(Q, G_n, k) = 1$ has cardinality $2k-1$. As a consequence of this consideration we have $W(I_n, G_n, k) \ge \frac{n-1}{2k-1}$ and then

$$\frac{W(I_n, G_n, k)}{\frac{1}{k}L(I, G)} \ge \frac{2k}{2k-1} - \frac{2k}{n(2k-1)}. \tag{1}$$

If $\alpha < \frac{2k}{2k-1}$ using Equation 1 with n large enough we get a contradiction. □

We end this section by considering a slightly different version of the standard wavelength assignment problem for optical rings in which the maximum length of input requests is at most $c < n$. In this framework it is possible to prove the following result.

Theorem 14. *If we consider requests of length at most c, then there exists k_c (which depends only on c) such that for every n-ring G we have $Gap(G, k_c) = 1$.*

Sketch of proof. Let I be any instance on any n-ring G. As usual we assume that for every edge e of G we have $L(I, G, e) = L(I, G)$. First we note that the number of distinct crossing paths (defined in the proof of Thm. 11) of I is at most c. As a consequence of this fact, we claim that it is possible to find a set of requests $I' \subseteq I$ such that

$$L(I', G) \le c \quad \text{and} \quad L(I \setminus I', G) = L(I, G) - L(I', G). \tag{2}$$

The set I' can be constructed as follows. We start from $I' = \{cp\}$ where cp is any crossing path. Then we move clockwise on the ring adding to I' a sequence of adjacent paths until we reach a new crossing path cp' whose first endpoint is equal to the first endpoint of another crossing path $cp'' \in I'$. Note that this must happen within at most c complete rounds since the number of distinct crossing paths is at most c. At this point we do not add cp' to I' but we remove from I' all the paths (possibly none) inserted before cp''. It takes a little effort to prove that I' satisfies properties (2). We repeat this procedure until all paths of I have been considered getting a partition of I into m subsets $I'_i,\ 1 \le i \le m$, such that

$$L(I'_i, G) \le c \quad \text{and} \quad \sum_{i=1}^{m} L(I'_i, G) = L(I, G). \tag{3}$$

It remains to be proven that there exists k_c such that using k_c fibres per link $Gap(S, G, k_c) = 1$ for every sequence of instances S. It can be proven (due to limited space we omit this part of the proof) that chosing $k_c = c!$ no sequence of instances can produce a gap bigger than 1. □

Acknowledgments

We would like to thank Amit Kumar and Bruno Codenotti for many useful discussions and Jon Kleinberg for having suggested the proof of Thm. 4.

References

1. R. P. Anstee. *An algorithmic proof of Tutte's f-factor theorem.* Journal of Algorithms, 6:112–131, 1985.
2. B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro. *Graph problems arising from wavelength-routing in all-optical networks.* Proc. of Workshop on Optics in Computer Science WOCS'97.
3. N. K. Cheung, N. K., and G. Winzer. *Special issue on dense WDM networks.* Journal on Selected Areas in Communications, 8, 1990.
4. T. Erlebach and K. Jansen. *Scheduling of virtual connections in fast networks.* In Proc. of Parallel Systems and Algorithms (PASA), 13–32, 1996.
5. T. Erlebach and K. Jansen. *Call scheduling in trees, rings and meshes.* In Proc. of HICSS, 1997.
6. M. C. Golumbic and R. E. Jamison. *The edge intersection graphs of paths in a tree.* Journal of Combinatorial Theory, Series B, 38:8–22, 1985.
7. P. E. Green. *Fibre–Optic Communication Networks.* Prentice–Hall, 1993.
8. I. Holyer. *The NP–completeness of edge coloring.* SIAM Journal of Computing, 10(4):718–720, 1981.
9. C. Kaklamanis, G. Persiano, T. Erlebach,and K. Jansen. *Constrained bipartite edge coloring with applications to wavelength routing.* Proc. of ICALP'97, Lecture notes in Computer Science vol. 1256:493–504, 1997.
10. J. Kleinberg and A. Kumar. *Wavelength conversion in optical networks* In Proc. 10th ACM-SIAM Symposium on Discrete Algorithms, 1999.
11. L. Lovász. *On chromatic number of finite set-systems.* Acta Math. Acad. Sci. Hungar, 19:59–67, 1968.
12. A. D. McAulay. *Optical Computer Architectures.* John Wiley, 1991.
13. J. Petersen. *Die Theorie der Regulären Graphen.* Acta Math. 15, 193–220, 1891.
14. R. Ramaswami. *Multiwavelength lightwave networks for computer communication.* IEEE Communications Magazine, 31(2):78–88, Feb. 1993.
15. R. E. Tarjan. *Decomposition by clique separators.* Discrete Mathematics, 55(2):221–232, 1985.
16. A. Tucker. *Coloring a family of circular arcs.* SIAM Journal of Applied Mathematics, 29(3):493–502, 1975.
17. R. J. Vetter and D. H. C. Du. *Distributed computing with high-speed optical networks.* IEEE Computer, 26(2):8–18, Feb. 1993.
18. G. Wilfong and P. Winkler. *Ring routing and wavelength translation.* In Proc. of the 9th Annual ACM-SIAM Symposium on on Discrete Algorithms (SODA), 333-341, 1998.

On the Logical Characterisation of Performability Properties

Christel Baier[a], Boudewijn Haverkort[b], Holger Hermanns[c], and Joost-Pieter Katoen[c]

[a]Institut für Informatik I, University of Bonn
Römerstraße 164, D-53117 Bonn, Germany

[b]Dept. of Computer Science, RWTH Aachen
Ahornstraße 55, D-52056 Aachen, Germany

[c]Dept. of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract. Markov-reward models, as extensions of continuous-time Markov chains, have received increased attention for the specification and evaluation of performance and dependability properties of systems. Until now, however, the specification of reward-based performance and dependability measures has been done manually and informally. In this paper, we change this undesirable situation by the introduction of a continuous-time, reward-based stochastic logic. We argue that this logic is adequate for expressing performability measures of a large variety. We isolate two important sub-logics, the logic **CSL** [1,3], and the novel logic **CRL** that allows one to express reward-based properties. These logics turn out to be complementary, which is formally established in our main duality theorem. This result implies that reward-based properties expressed in **CRL** for a particular Markov reward model can be interpreted as **CSL** properties over a derived continuous-time Markov chain, so that model checking procedures for **CSL** [3,2] can be employed.

1 Introduction

With the advent of fault-tolerant and distributed computer and communication systems, the classical separation between performance evaluation and dependability (i.e., reliability, availability and timeliness) evaluation does not make sense anymore. Instead, the combined performance and dependability of a system is of critical importance. This observation led to development of the performability evaluation framework [12,13]. This framework allows one to specify models that include both performance-related and dependability-related events in a natural way. Furthermore, the choice of Markov-reward models (MRMs) [11] as mathematical basis allows one to specify a wide variety of measures of interest, albeit at times in a slightly cumbersome way. An MRM is a continuous-time Markov chain (CTMC) augmented with a *reward structure* assigning a real-valued reward to each state in the model. Such reward can be interpreted as bonus, gain, or dually, as cost. Typical measures of interest express the amount of gain accumulated by the system, over a finite or infinite time-horizon.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 780–792, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Given the fact that the model is stochastic, the measures of interest are stochastic variables. MRMs have shown to pair a reasonable modelling flexibility and expressiveness with manageable computational expenses for the model evaluation. To increase the modelling flexibility, a number of application-oriented model specification techniques and supporting tools have been developed [8].

The specification of the measure-of-interest for a given MRM can not always be done conveniently, nor can all possible measures-of-interest be expressed conveniently. In particular, until recently it has not been possible to directly express measures where state *sequences* or paths matter, nor to accumulate rewards only in certain subsets of states, if the rewards outside these subsets are non-zero. Such measures are then either "specified" informally, with all its negative implications, or require a manual tailoring of the model so as to address the right subsets of states. An example of a measure that is very difficult to specify directly is the expected amount of gain obtained from the system until a particular state is reached, provided that all paths to that state obey certain properties.

Recently, Obal and Sanders have proposed a technique to specify so-called path-based reward variables [14] by which the specification of measures over state sequences becomes more convenient, because it avoids the manual tailoring of the model. In the context of the stochastic process algebra PEPA, Clark *et al.* recently proposed the use of a probabilistic modal logic to ease the specification of reward *structures* of MRM [5], as opposed to the specification of reward-based *measures*, as we do.

In [3] we proposed to specify measures of interest for CTMCs in the logic **CSL** (Continuous Stochastic Logic), a superset of the (equally named) logic introduced by Aziz *et al.* [1]. **CSL** includes a timed **CTL**-like time-bounded until operator, and a steady-state operator. Using this logic, very complex measures can be expressed easily; model-checking algorithms for **CSL** have been proposed [3,2] (and implemented [10]). Notice however, that **CSL** is interpreted over CTMCs only, and is hence not able to address reward-based measures. The current paper extends this work, in that Markov-*reward* models are evaluated, i.e., CTMCs augmented with a reward structure.

In this paper, we introduce a novel continuous-time, stochastic reward-based logic **CSRL**, that is adequate for expressing performability measures of a large variety. It includes next and until operators, that are equipped with time-interval- as well as reward-interval-bounds. We present syntax and formal semantics of the logic, and isolate two important sub-logics: the logic **CSL**, and the logic **CRL** (Continuous Reward Logic) that allows one to express time-independent reward-based properties. These logics turn out to be complementary, which is formally established in a main duality theorem, showing that time- and reward-intervals are interchangeable. More precisely, we show that for each MRM $\mathcal{M}$ and formula Φ the set of states satisfying Φ equals the set of states of a derived MRM $\mathcal{M}^{-1}$ satisfying formula Φ^{-1}, where the latter is obtained from Φ by simply swapping time- and reward-intervals. The transformation of $\mathcal{M}$ is inspired by [4]. The fixpoint characterisations for the **CRL** path operators (interpreted over an MRM) reduce to the characterisations that are used for model

checking **CSL** (over a CTMC). As a consequence of the duality result, the model checking problem for **CRL** is reducible to the model checking problem for **CSL** and hence solvable with existing techniques for **CSL**.

The paper is organised as follows. Section 2 introduces MRMs and typical measures of interest for them. In Section 3 the logic **CSRL** and its sub-logics are defined, whereas Section 4 presents the main duality theorem. Section 5 discusses its consequences for model checking and highlights that most reward-based performability measures having appeared in the literature can be expressed as simple formulas of (a minor extension of) the logic. Section 6 concludes the paper.

2 Markov Reward Models

In this section we introduce the basic concepts of MRMs [11]. We slightly depart from the standard notation for MRMs (and CTMCs) and consider an MRM as an ordinary transition system, i.e., a Kripke structure, where the edges are equipped with probabilistic timing information and the states are augmented with a real number that indicates the earned reward per unit of time for staying in a state. This then allows the usual interpretation of linear-time temporal operators like next step and unbounded or time-bounded until.

MRMs. Let AP be a fixed, finite set of atomic propositions.

Definition 1. *A (labelled) CTMC $\mathcal{C}$ is a tuple $(S, \mathbf{R}, L)$ where S is a finite set of* states*, $\mathbf{R} : S \times S \to I\!R_{\geqslant 0}$ the* rate matrix*, and $L : S \to 2^{AP}$ the* labelling *function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions $a \in AP$ that are valid in s. A state s is called* terminal *(or* absorbing*) iff $\mathbf{R}(s, s') = 0$ for all states s'.*

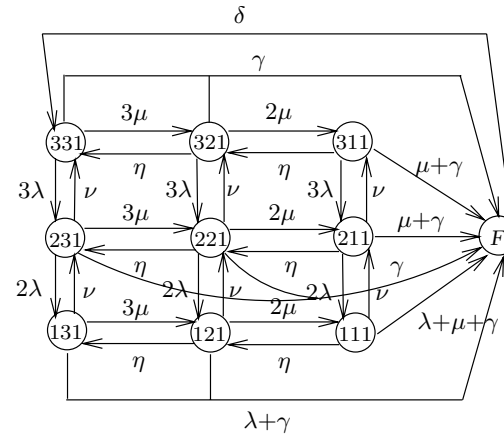
Intuitively, $\mathbf{R}(s, s') > 0$ iff there is a transition from s to s'; $1 - e^{-\mathbf{R}(s,s')\cdot t}$ is the probability that the transition $s \to s'$ can be triggered within t time units. Thus the delay of transition $s \to s'$ is governed by an exponential distribution with rate $\mathbf{R}(s, s')$. If $\mathbf{R}(s, s') > 0$ for more than one state s', a competition between the transitions exists, known as the *race condition*. The probability to move from non-absorbing s to s' within t time units, i.e., $s \to s'$ to win the race, is given by

$$\frac{\mathbf{R}(s, s')}{\mathbf{E}(s)} \cdot \left(1 - e^{-\mathbf{E}(s)\cdot t}\right)$$

where $\mathbf{E}(s) = \sum_{s' \in S} \mathbf{R}(s, s')$ denotes the *total rate* at which any transition emanating from state s is taken. More precisely, $\mathbf{E}(s)$ specifies that the probability of leaving s within t time-units is $1 - e^{-\mathbf{E}(s)\cdot t}$, because the minimum of exponential distributions, competing in a race, is characterised by the sum of their rates. Consequently, the probability of moving from a non-absorbing state s to s' by a single transition, denoted $\mathbf{P}(s, s')$, is determined by the probability that the delay of moving from s to s' finishes before the delays of other outgoing edges from s; formally, $\mathbf{P}(s, s') = \mathbf{R}(s, s')/\mathbf{E}(s)$. For absorbing states, the total rate $\mathbf{E}(s) = 0$; we then have $\mathbf{P}(s, s') = 0$ for any state s'.

Definition 2. *A (labelled) MRM $\mathcal{M}$ is a pair $(\mathcal{C}, \rho)$ where $\mathcal{C}$ is a (labelled) CTMC, and $\rho : S \to \mathbb{R}_{\geqslant 0}$ is a* reward structure *that assigns to each state $s \in S$ a reward $\rho(s)$, also called gain or bonus or dually, cost.*

Example 1. As a running example we consider a fault-tolerant multiprocessor system inspired by [15]. The system consists of three processors, three memories, and a single interconnection network that allows a processor to access any memory. We model this system by a CTMC, depicted below, where state $(i, j, 1)$ models that i processors and j memories $(1 \leqslant i, j < 4)$ are operational and are connected by a single network. Initially all components are functioning correctly, i.e., the initial state is $(3, 3, 1)$. The minimal operational configuration of the system is $(1, 1, 1)$. The failure rate of a processor is λ, of a memory μ, and of the network γ failures per hour (fph). It is assumed that a single repair unit is present to repair all types of components. The expected repair time of a processor is $1/\nu$ and of a memory $1/\eta$ hours. In case all memories, all processors, or the network has failed the system moves to state F. After a repair in state F, we assume the system to restart in state $(3, 3, 1)$ with rate δ.



The reward structure can be instantiated in different ways so as to specify a variety of performability measures. The following reward structures are taken from [15]. The simplest reward structure (leading to an availability model) divides the states into operational and non-operational states: $\rho_1(F) = 0$ and $\rho_1(i, j, k) = 1$. A reward structure in which varying levels of performance of the system are represented is for instance based on the capacity of the system: $\rho_2(F) = 0$ and $\rho_2(i, j, k) = min(i, j)$. The third reward structure does consider processors contending for the memories, by taking as reward for operational states the expected available memory bandwidth: $\rho_3(F) = 0$ and $\rho_3(i, j, k) = m \cdot \big(1 - (1 - 1/m)^l\big)$ where $l = min(i, j)$ and $m = max(i, j)$. ■

Let $\mathcal{M} = (\mathcal{C}, \rho)$ be an MRM with underlying CTMC $\mathcal{C} = (S, \mathbf{R}, L)$.

Paths. An infinite *path* σ is a sequence $s_0, t_0, s_1, t_1, s_2, t_2, \ldots$ with for $i \in \mathbb{N}$, $s_i \in S$ and $t_i \in \mathbb{R}_{>0}$ such that $\mathbf{R}(s_i, s_{i+1}) > 0$. For $i \in \mathbb{N}$ let $\sigma[i] = s_i$, the $(i{+}1)$-st state of σ, and $\delta(\sigma, i) = t_i$, the time spent in s_i. For $t \in \mathbb{R}_{\geqslant 0}$ and i the smallest index with $t \leqslant \sum_{j=0}^{i} t_j$ let $\sigma@t = \sigma[i]$, the state in σ at time t. For $t = \sum_{j=0}^{k-1} t_j + t'$ with $t' \leqslant t_k$ we define $y(\sigma, t) = \sum_{j=0}^{k-1} t_j \cdot \rho(s_j) + t' \cdot \rho(s_k)$, the cumulative reward along σ up to time t.

A finite path σ is a sequence $s_0, t_0, s_1, t_1, s_2, t_2, \ldots, t_{l-1}, s_l$ where s_l is absorbing, and $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $i < l$. For finite σ, $\sigma[i]$ and $\delta(\sigma, i)$ are only defined for $i \leqslant l$; they are defined as above for $i < l$, and $\delta(\sigma, l) = \infty$. For $t > \sum_{j=0}^{l-1} t_j$ we let $\sigma@t = s_l$ and let the cumulative reward $y(\sigma, t) = \sum_{j=0}^{l-1} t_j \cdot \rho(s_j) + (t - \sum_{j=0}^{l-1} t_j) \cdot \rho(s_l)$; for the other cases, $\sigma@t$ and $y(\sigma, t)$ are defined as above.

Let $Path^{\mathcal{M}}(s)$ denote the set of (finite and infinite) paths starting in s.

Borel space. Any state $s = s_0$ yields a probability measure Pr on $Path^{\mathcal{M}}(s)$ as follows. Let $s_0, \ldots, s_k \in S$ with $\mathbf{R}(s_i, s_{i+1}) > 0$, $(0 \leqslant i < k)$, and $I_0, \ldots, I_{k-1}$ non-empty intervals in $\mathbb{R}_{\geqslant 0}$. Then, $C(s_0, I_0, \ldots, I_{k-1}, s_k)$ denotes the *cylinder set* consisting of all paths $\sigma \in Path^{\mathcal{M}}(s)$ such that $\sigma[i] = s_i$ $(i \leqslant k)$, and $\delta(\sigma, i) \in I_i$ $(i < k)$. Let $\mathcal{F}(Path^{\mathcal{M}}(s))$ be the smallest σ-algebra on $Path^{\mathcal{M}}(s)$ which contains all sets $C(s, I_0, \ldots, I_{k-1}, s_k)$ where $s_0, \ldots, s_k$ ranges over all state-sequences with $s = s_0$, $\mathbf{R}(s_i, s_{i+1}) > 0$ $(0 \leqslant i < k)$, and $I_0, \ldots, I_{k-1}$ ranges over all sequences of non-empty intervals in $\mathbb{R}_{\geqslant 0}$. The probability measure Pr on $\mathcal{F}(Path^{\mathcal{M}}(s))$ is the unique measure defined by induction on k: $\Pr(C(s_0)) = 1$, and for $k \geqslant 0$,

$$\Pr(C(s_0, \ldots, s_k, I', s')) = \Pr(C(s_0, \ldots, s_k)) \cdot \mathbf{P}(s_k, s') \cdot \left(e^{-\mathbf{E}(s_k) \cdot a} - e^{-\mathbf{E}(s_k) \cdot b}\right),$$

where $a = \inf I'$ and $b = \sup I'$. (For $b = \infty$ and $\lambda > 0$ let $e^{-\lambda \cdot \infty} = 0$.) Note that $e^{-\mathbf{E}(s_k) \cdot a} - e^{-\mathbf{E}(s_k) \cdot b}$ is the probability of leaving state s_k in the interval I'.

Remark. For infinite paths we do not assume *time divergence*. Although such paths represent "unrealistic" computations where infinitely many transitions are taken in a finite amount of time, the probability measure of such Zeno paths is 0. This justifies a lazy treatment of the notations $\sigma@t$ and $y(\sigma, t)$ when we refer to the probability of a measurable ste of paths. ■

Steady-state and transient probabilities. For a CTMC $\mathcal{C}$ two major types of state probabilities are distinguished: steady-state probabilities where the system is considered "on the long run", i.e., when an equilibrium has been reached, and transient probabilities where the system is considered at a given time instant t. Formally, the *transient probability*

$$\pi^{\mathcal{C}}(s, s', t) \;=\; \Pr\{\sigma \in Path^{\mathcal{C}}(s) \mid \sigma@t = s'\}$$

stands for the probability to be in state s' at time t given the initial state s. Note that this set is measurable. *Steady-state probabilities* are defined as

$$\pi^{\mathcal{C}}(s, s') = \lim_{t \to \infty} \pi^{\mathcal{C}}(s, s', t).$$

This limit always exists for finite CTMCs. For $S' \subseteq S$, $\pi^{\mathcal{C}}(s, S') = \sum_{s' \in S'} \pi^{\mathcal{C}}(s, s')$ denotes the steady-state probability for set S'. In the sequel, we will often use $\mathcal{M}$ rather than $\mathcal{C}$ (the underlying CTMC of $\mathcal{M}$) as superscript.

3 Stochastic CTL with Time and Rewards

This section introduces a stochastic logic to reason about reward-based as well as time-based constraints, and identifies two important sub-logics of it. For explanatory purposes, we first introduce a simple branching time logic without any support for real time or reward constraints.

Basic logic. The base stochastic logic **SL**, a stochastic variant of **CTL** (Computational Tree Logic), is a continuous-time variant of **PCTL** [7].

Syntax. For $a \in AP$, $p \in [0,1]$ and $\bowtie \in \{\leqslant, <, \geqslant, >\}$, the state-formulas of **SL** are defined by the grammar

$$\Phi ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \Phi \wedge \Phi \;\Big|\; \neg\Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$

where path-formulas are defined by $\varphi ::= X\Phi \;\Big|\; \Phi\,\mathcal{U}\,\Phi$.

Other boolean connectives such as $\vee$ and $\rightarrow$ are derived in the obvious way. As usual $\Diamond\Phi = \mathrm{tt}\,\mathcal{U}\,\Phi$ and the $\Box$-operator can be obtained by, for example, $\mathcal{P}_{\geqslant p}(\Box\Phi) = \neg\mathcal{P}_{\geqslant 1-p}(\Diamond\neg\Phi)$. The state-formula $\mathcal{S}_{\bowtie p}(\Phi)$ asserts that the steady-state probability for the set of Φ-states meets the bound $\bowtie p$. For the running example, the formula $\mathcal{S}_{\geqslant 0.8}(\mathit{2pup})$ expresses that the steady-state probability to be in a state with two operational processors is at least 0.8 where *2pup* holds in state $(2, j, 1)$, $1 \leqslant j < 4$. The operator $\mathcal{P}_{\bowtie p}(.)$ replaces the usual **CTL** path quantifiers $\exists$ and $\forall$. $\mathcal{P}_{\bowtie p}(\varphi)$ asserts that the probability measure of the paths satisfying φ meets the bound $\bowtie p$. For example, $\mathcal{P}_{\geqslant 0.3}(\Diamond F)$ denotes that the probability to eventually reach the failure state of the multi-processor system is at least 0.3.

Semantics. The **SL** state-formulas are interpreted over the states of a CTMC $\mathcal{C} = (S, \mathbf{R}, L)$ (or an MRM $\mathcal{M}$ with underlying CTMC $\mathcal{C}$) with proposition labels in AP. Let $Sat^{\mathcal{C}}(\Phi) = \{\, s \in S \mid s \models \Phi \,\}$.

$$\begin{array}{llll}
s \models \mathrm{tt} & \text{for all } s \in S & s \models \Phi_1 \wedge \Phi_2 & \text{iff } s \models \Phi_i, \text{ for } i{=}1,2 \\
s \models a & \text{iff } a \in L(s) & s \models \mathcal{S}_{\bowtie p}(\Phi) & \text{iff } \pi^{\mathcal{C}}(s, Sat^{\mathcal{C}}(\Phi)) \bowtie p \\
s \models \neg\Phi & \text{iff } s \not\models \Phi & s \models \mathcal{P}_{\bowtie p}(\varphi) & \text{iff } Prob^{\mathcal{C}}(s, \varphi) \bowtie p
\end{array}$$

Here, $Prob^{\mathcal{C}}(s, \varphi)$ denotes the probability measure of all paths satisfying φ given that the system starts in state s, i.e.,

$$Prob^{\mathcal{C}}(s, \varphi) \;=\; \Pr\{\, \sigma \in Path^{\mathcal{C}}(s) \mid \sigma \models \varphi \,\}.$$

The fact that the set $\{\, \sigma \in Path^{\mathcal{C}}(s) \mid \sigma \models \varphi \,\}$ is measurable can be easily verified. The intended meaning of the temporal operators $\mathcal{U}$ and X is standard:

$$\begin{array}{ll}
\sigma \models X\Phi & \text{iff } \sigma[1] \text{ is defined and } \sigma[1] \models \Phi \\
\sigma \models \Phi_1\,\mathcal{U}\,\Phi_2 & \text{iff } \exists k \geqslant 0.\, (\sigma[k] \models \Phi_2 \wedge \forall 0 \leqslant i < k.\, \sigma[i] \models \Phi_1).
\end{array}$$

Alternative characterisations. For next-formulas we have, as for DTMCs [7]:

$$Prob^{\mathcal{C}}(s, X\Phi) = \mathbf{P}(s, \Phi) \tag{1}$$

where $\mathbf{P}(s,\Phi) = \sum_{s' \in Sat^{\mathcal{C}}(\Phi)} \mathbf{P}(s,s')$, the probability to reach a Φ-state in one step from s. For until-formulas we have that the probability $Prob^{\mathcal{C}}(s, \Phi_1 \mathcal{U} \Phi_2)$ is the least solution[1] of the following set of equations: $Prob^{\mathcal{C}}(s, \Phi_1 \mathcal{U} \Phi_2)$ equals 1 if $s \models \Phi_2$, equals

$$\sum_{s' \in S} \mathbf{P}(s,s') \cdot Prob^{\mathcal{C}}(s', \Phi_1 \mathcal{U} \Phi_2) \tag{2}$$

if $s \models \Phi_1 \wedge \neg\Phi_2$, and 0 otherwise. This probability can be computed as the solution of a regular system of linear equations by standard means such as Gaussian elimination [6] or can be approximated by an iterative approach.

The full logic. We now extend **SL** by providing means to reason about both time constraints and cumulative reward constraints. We refer to this logic as **CSRL**. Later we will identify fragments of **CSRL** that refer to only time, respectively only reward constraints.

Syntax. The syntax (and semantics) of the state formulas of **CSRL** are defined as for the basic logic. Path-formulas φ are defined for intervals $I, J \subseteq \mathbb{R}_{\geqslant 0}$ by:

$$\varphi \ ::= \ X_J^I \Phi \ \Big| \ \Phi \mathcal{U}_J^I \Phi.$$

In a similar way as before, we define $\Diamond_J^I \Phi = \mathrm{tt}\, \mathcal{U}_J^I \Phi$ and $\mathcal{P}_{\bowtie p}(\Box_J^I \Phi) = \neg \mathcal{P}_{\bowtie p}(\Diamond_J^I \neg \Phi)$. Interval I can be considered as a timing constraint whereas J represents a bound for the cumulative reward. The path-formula $X_J^I \Phi$ asserts that a transition is made to a Φ-state at time point $t \in I$ such that the earned cumulative reward r until time t meets the bounds specified by J, i.e., $r \in J$. The semantics of $\Phi_1 \mathcal{U}_J^I \Phi_2$ is as for $\Phi_1 \mathcal{U} \Phi_2$ with the additional constraints that the Φ_2-state is reached at some time point t in I and the earned cumulative reward up to t lies in J. As an example property for the multi-processor system, $\mathcal{P}_{\geqslant 0.95}(\Diamond_{[0,2]}^{[60,60]} \mathrm{tt})$ denotes that with probability at least 0.95 the cumulative reward (e.g., the expected capacity of the system for reward structure ρ_2) at time instant 60 is at most 2. Given that the reward of a state indicates the number of jobs processed per time-unit, property $\mathcal{P}_{\geqslant 0.98}(\mathit{3mup}\, \mathcal{U}_{[7,\infty)}^{[0,30]}\, \mathit{mdown})$ expresses that with probability at least 0.98 at least 7 jobs have been processed (starting from the initial state) before the first memory unit fails within 30 time units, where *3mup* is valid in states $(i,3,1)$, $1 \leqslant i < 4$ and *mdown* is valid in states $(i,2,1)$, $0 \leqslant i < 4$.

Semantics. The semantics of the **CSRL** path-formulas is defined as follows:

$$\begin{array}{ll} \sigma \models X_J^I \Phi & \text{iff } \sigma[1] \text{ is defined and } \sigma[1] \models \Phi \wedge \delta(\sigma,0) \in I \wedge y(\sigma, \delta(\sigma,0)) \in J \\ \sigma \models \Phi_1 \mathcal{U}_J^I \Phi_2 & \text{iff } \exists t \in I.\, (\sigma @ t \models \Phi_2 \wedge (\forall t' \in [0,t).\, \sigma @ t' \models \Phi_1) \wedge y(\sigma,t) \in J). \end{array}$$

Special cases occur for $I = [0,\infty)$ and $J = [0,\infty)$:

$$X\Phi = X_{[0,\infty)}^{[0,\infty)} \Phi \ \text{ and } \Phi_1 \mathcal{U} \Phi_2 = \Phi_1 \mathcal{U}_{[0,\infty)}^{[0,\infty)} \Phi_2.$$

[1] Strictly speaking, the function $s \mapsto Prob^{\mathcal{C}}(s, \Phi_1 \mathcal{U} \Phi_2)$ is the least fixpoint of a higher-order function on $(S \to [0,1]) \to (S \to [0,1])$ where the underlying partial order on $S \to [0,1]$ is defined for $F_1, F_2 : S \to [0,1]$ by $F_1 \leqslant F_2$ iff $F_1(s) \leqslant F_2(s)$ for all $s \in S$.

Thus, **SL** is a proper subset of this logic. The logic **CSL** [1,3] (or, timed stochastic CTL) is obtained in case $J = [0, \infty)$ for all sub-formulas. Similarly, we obtain the new logic **CRL** (reward-based stochastic CTL) in case $I = [0, \infty)$ for all sub-formulas. In the sequel, intervals of the form $[0, \infty)$ are often omitted from the operators.

We recall that $y(\sigma, t)$ denotes the cumulative reward along the prefix of σ up to time t. The intuition behind $y(\sigma, t)$ depends on the formula under consideration and the interpretation of the rewards in the MRM $\mathcal{M}$ under consideration. For instance, for $\varphi = \Diamond good$ and path σ that satisfies φ, the cumulative reward $y(\sigma, t)$ can be interpreted as the cost to reach a *good* state within t time units. For $\varphi = \Diamond bad$, it may be interpreted as the gain earned before reaching a *bad* state within t time units.

Alternative characterisations. We first observe that it suffices to consider time and reward bounds specified by closed intervals. Let $K = \{\, x \in I \mid \rho(s) \cdot x \in J \,\}$ for closed intervals I and J. The probability of leaving state s at some time point x within the interval I such that the earned reward $\rho(s) \cdot x$ lies in J is can be expressed by

$$\mathbf{P}^I_J(s) = \int_K \mathbf{E}(s) \cdot e^{-\mathbf{E}(s) \cdot x} \, dx.$$

For instance, $\mathbf{P}^{[0,t]}_{[0,\infty)}(s) = 1 - e^{-\mathbf{E}(s) \cdot t}$, the probability to leave state s within t time units where the reward earned is irrelevant. If $\rho(s) = 2$, $I = [1, 3]$ and $J = [9, 11]$ then $K = \varnothing$ and $\mathbf{P}^I_J(s) = 0$. For $X^I_J \Phi$ we obtain:

$$Prob^{\mathcal{M}}(s, X^I_J \Phi) = \mathbf{P}^I_J(s) \cdot \mathbf{P}(s, \Phi).$$

For the case $I = J = [0, \infty)$ this reduces to equation (1).

Let $I \ominus x$ denote $\{\, t - x \mid t \in I, t \geqslant x \,\}$. For $\varphi = \Phi_1 \, \mathcal{U}^I_J \, \Phi_2$ we have that $Prob^{\mathcal{M}}(s, \varphi)$ is the least solution of the following set of equations: $Prob^{\mathcal{M}}(s, \varphi) = 1$ if $s \models \neg\Phi_1 \wedge \Phi_2$, $\inf I = 0$ and $\inf J = 0$,

$$\int_0^{\sup K} \sum_{s' \in S} \mathbf{P}(s, s', x) \cdot Prob^{\mathcal{M}}(s', \Phi_1 \, \mathcal{U}^{I \ominus x}_{J \ominus \rho(s) \cdot x} \, \Phi_2) \, dx \tag{3}$$

if $s \models \Phi_1 \wedge \neg\Phi_2$, and

$$e^{-\mathbf{E}(s) \cdot \inf K} + \int_0^{\inf K} \sum_{s' \in S} \mathbf{P}(s, s', x) \cdot Prob^{\mathcal{M}}(s', \Phi_1 \, \mathcal{U}^{I \ominus x}_{J \ominus \rho(s) \cdot x} \, \Phi_2) \, dx$$

if $s \models \Phi_1 \wedge \Phi_2$, and 0 otherwise, where $\mathbf{P}(s, s', x) = \mathbf{R}(s, s') \cdot e^{-\mathbf{E}(s) \cdot x}$ denotes the probability of moving from state s to s' within x time units. The above characterisation is justified as follows. If s satisfies Φ_1 and $\neg\Phi_2$, the probability of reaching a Φ_2-state from s within the interval I by earning a reward $r \in J$ equals the probability of reaching some direct successor s' of s within x time units ($x \leqslant \sup I$ and $\rho(s) \cdot x \leqslant \sup J$, that is, $x \leqslant \sup K$), multiplied by the probability of reaching a Φ_2-state from s' in the remaining time interval $I \ominus x$

while earning a reward of $r-\rho(s)\cdot x$. If s satisfies $\Phi_1 \wedge \Phi_2$, the path-formula φ is satisfied if no transition outgoing from s is taken for at least $\inf K$ time units (first summand).[2] Alternatively, state s should be left before $\inf K$ in which case the probability is defined in a similar way as for the case $s \models \Phi_1 \wedge \neg\Phi_2$ (second summand). Note that $\inf K = 0$ is possible (if e.g., $\inf J = \inf I = 0$). In this case, $s \models \Phi_1 \wedge \Phi_2$ yields that any path starting in s satisfies $\Phi_1 \mathcal{U}_J^I \Phi_2$ and $Prob^{\mathcal{M}}(s, \Phi_1 \mathcal{U}_J^I \Phi_2) = 1$.

If the reward constraint is trivial, i.e., $J = [0,\infty)$, and I is of the form $[0,t]$ for $t \in \mathbb{R}_{\geqslant 0}$, then the characterisation for $\mathcal{U}^I$ reduces to the least solution of the following set of equations: $Prob^{\mathcal{M}}(s, \Phi_1 \mathcal{U}^{[0,t]} \Phi_2)$ equals 1 if $s \models \Phi_2$, equals

$$\int_0^t \sum_{s' \in S} \mathbf{P}(s,s',x) \cdot Prob^{\mathcal{M}}(s', \Phi_1 \mathcal{U}^{[0,t-x]} \Phi_2)\, dx \tag{4}$$

if $s \models \Phi_1 \wedge \neg\Phi_2$, and 0 otherwise. This coincides with the characterisation for time-bounded until in [3]. For the special case $I = J = [0,\infty)$ we obtain $K = [0,\infty)$ and hence the characterisation for $\mathcal{U}^I$ reduces to (2).

4 Duality

In this section we present the main result of the paper, a duality theorem that has important consequences for model checking sub-logics of **CSRL**. The basic idea behind this duality, inspired by [4], is that the progress of time can be regarded as the earning of reward and vice versa. First we obtain a duality result for MRMs where all states have a positive reward. After that we consider the (restricted) applicability of the duality result to MRMs with zero rewards.

Transformation of MRMs. Let $\mathcal{M} = (S, \mathbf{R}, L, \rho)$ be an MRM that satisfies $\rho(s) > 0$ for any state s. Define MRM $\mathcal{M}^{-1} = (S, \mathbf{R}', L, \rho')$ that results from $\mathcal{M}$ by: (i) rescaling the transition rates by the reward of their originating state (as originally proposed in [4]), i.e., $\mathbf{R}'(s,s') = \mathbf{R}(s,s')/\rho(s)$ and, (ii) inverting the reward structure, i.e., $\rho'(s) = 1/\rho(s)$. Intuitively, the transformation of $\mathcal{M}$ into $\mathcal{M}^{-1}$ stretches the residence time in state s with a factor that is proportional to the reciprocal of its reward $\rho(s)$ if $\rho(s) > 1$, and it compresses the residence time by the same factor if $0 < \rho(s) < 1$. The reward structure is changed similarly. Note that $\mathcal{M} = (\mathcal{M}^{-1})^{-1}$.

One might interpret the residence of t time units in $\mathcal{M}^{-1}$ as the earning of t reward in state s in $\mathcal{M}$, or (reversely) an earning of a reward r in state s in $\mathcal{M}$ corresponds to a residence of r in $\mathcal{M}^{-1}$. Thus, the notions of time and reward in $\mathcal{M}$ are reversed in $\mathcal{M}^{-1}$. Accordingly:

Lemma 1. *For MRM $\mathcal{M} = (S, \mathbf{R}, L, \rho)$ with $\rho(s) > 0$ for all $s \in S$ and* **CSRL** *state-formulas Φ, Φ_1 and Φ_2:*

1. $Prob^{\mathcal{M}}(s, X_J^I \Phi) = Prob^{\mathcal{M}^{-1}}(s, X_I^J \Phi)$

[2] By convention, $\inf \varnothing = \infty$.

2. $Prob^{\mathcal{M}}(s, \Phi_1 \, \mathcal{U}_J^I \, \Phi_2) = Prob^{\mathcal{M}^{-1}}(s, \Phi_1 \, \mathcal{U}_I^J \, \Phi_2)$.

We informally justify *2.* for $I = [0,t]$ and $J = [0,r]$ with $r, t \in \mathbb{R}_{\geqslant 0}$. Let MRM $\mathcal{M} = (S, \mathbf{R}, L, \rho)$ with $\rho(s) > 0$ for all $s \in S$. Let $s \in S$ be such that $s \models \Phi_1 \wedge \neg\Phi_2$. From equation (3) we have that $Prob^{\mathcal{M}^{-1}}(s, \Phi_1 \, \mathcal{U}_I^J \, \Phi_2)$ equals

$$\int_{K'} \sum_{s' \in S} \mathbf{P}(s, s', x) \cdot Prob^{\mathcal{M}^{-1}}(s', \Phi_1 \, \mathcal{U}_{I \ominus \rho'(s) \cdot x}^{J \ominus x} \, \Phi_2) \; dx.$$

for $K' = \{ \, x \in [0,t] | \rho'(s) \cdot x \in [0,r] \, \}$, i.e., $K' = [0, min(t, \frac{r}{\rho'(s)})]$. By the definition of $\mathcal{M}^{-1}$ this equals

$$\int_{K'} \sum_{s' \in S} \frac{\mathbf{R}(s, s')}{\rho(s)} \cdot e^{-\frac{\mathbf{E}(s)}{\rho(s)} \cdot x} \cdot Prob^{\mathcal{M}^{-1}}(s', \Phi_1 \, \mathcal{U}_{I \ominus \frac{x}{\rho(s)}}^{J \ominus x} \, \Phi_2) \; dx.$$

By substitution $y = \frac{x}{\rho(s)}$ this integral reduces to:

$$\int_{K} \sum_{s' \in S} \mathbf{R}(s, s') \cdot e^{-\mathbf{E}(s) \cdot y} \cdot Prob^{\mathcal{M}^{-1}}(s', \Phi_1 \, \mathcal{U}_{I \ominus y}^{J \ominus \rho(s) \cdot y} \, \Phi_2) \; dy$$

where $K = [0, min(\frac{t}{\rho(s)}, r)]$. Thus, the function that maps (s, I, J) onto $Prob^{\mathcal{M}^{-1}}(s, \Phi_1 \, \mathcal{U}_I^J \, \Phi_2)$ meets the fixed point equation for $Prob^{\mathcal{M}}(s, \Phi_1 \, \mathcal{U}_J^I \, \Phi_2)$. Using arguments of fixed point theory, i.e., Tarski's theorem for least fixed points of monotonic functions on lattices, it can be shown that these fixed points agree (as they both are the least fixed point of the same operator). Thus, we obtain

$$\int_{K} \sum_{s' \in S} \mathbf{P}(s, s', y) \cdot Prob^{\mathcal{M}}(s', \Phi_1 \, \mathcal{U}_{J \ominus \rho(s) \cdot y}^{I \ominus y} \, \Phi_2) \; dy$$

and this equals $Prob^{\mathcal{M}}(s, \Phi_1 \, \mathcal{U}_J^I \, \Phi_2)$ for $s \models \Phi_1 \wedge \neg\Phi_2$, cf. (3).

For **CSRL** state-formula Φ let Φ^{-1} be defined as Φ where for each subformula in Φ of the form X_J^I or $\mathcal{U}_J^I$ the intervals I and J are swapped. This notion can be easily defined by structural induction on Φ and its definition is omitted here. For instance, for $\Phi = \mathcal{P}_{\geqslant 0.9}(\neg F \mathcal{U}_{[10,\infty)}^{[50,50]} F)$ we have $\Phi^{-1} = \mathcal{P}_{\geqslant 0.9}(\neg F \mathcal{U}_{[50,50]}^{[10,\infty)} F)$. We now have:

Theorem 1. *For MRM* $\mathcal{M} = (S, \mathbf{R}, L, \rho)$ *with* $\rho(s) > 0$ *for all* $s \in S$ *and* **CSRL** *state-formula* Φ*:*

$$Sat^{\mathcal{M}}(\Phi) \quad = \quad Sat^{\mathcal{M}^{-1}}(\Phi^{-1}).$$

If $\mathcal{M}$ contains states equipped with a zero reward, this duality result does not hold, as the reverse of earning a zero reward in $\mathcal{M}$ when considering Φ should correspond to a residence of 0 time units in $\mathcal{M}^{-1}$ for Φ^{-1}, which — as the advance of time in a state cannot be halted — is in general not possible. However, the result of Theorem 1 applies to some restricted, though still practical, cases, viz.

if (i) for each sub-formula of Φ of the form $X_J^I \Phi'$ we have $J = [0, \infty)$, and (ii) for each sub-formula of the form $\Phi_1 \mathcal{U}_J^I \Phi_2$ we either have $J = [0, \infty)$ or $Sat^{\mathcal{M}}(\Phi_1) \subseteq \{ s \in S \mid \rho(s) > 0 \}$, i.e., all Φ_1-states are positively rewarded. The intuition is that either the reward constraint (i.e., time constraint) is trivial in Φ (in Φ^{-1}), or that zero-rewarded states are not involved in checking the reward constraint. Here, we define $\mathcal{M}^{-1}$ by setting $\mathbf{R}'(s, s') = \mathbf{R}(s, s')$ and $\rho'(s) = 0$ in case $\rho(s) = 0$ and as defined above otherwise. For instance, Theorem 1 applies to the property $\mathcal{P}_{\geqslant 0.9}(\neg F \mathcal{U}_{[10,\infty)}^{[50,50]} F)$ for the multi-processor example, since all $\neg F$-states have a positive reward.

5 Application of the Logic

In this section, we discuss model checking of **CSRL**. We furthermore illustrate that **CSRL** and its fragments **CSL** and **CRL** provide ample means for the specification of performability measures.

Model checking. **CSL** model checking can be carried out in the following way. $\mathcal{S}_{\bowtie p}(\Phi)$ gives rise to a system of linear equations for each bottom strongly connected component of the graph underlying the CTMC [3]. The probability to satisfy $\mathcal{U}$- and X-path formulas can be obtained as the solution of a system of linear equations, resp. a single matrix-vector multiplication [7], based on (1) and (2). Finally, the probability to satisfy a $\mathcal{U}^I$-formula can be obtained as the solution of a system of Volterra integral equations (4), that can be computed by either numerical integration [3], or transient analysis of the CTMC [2]. From Theorem 1, we can conclude that model checking an MRM against a **CRL**-formula can be performed using the algorithms established for model checking CTMCs against **CSL**:

Corollary 1. *For an MRM without any zero rewards, model checking* **CRL** *is reducible to model checking* **CSL**.

In a number of interesting, albeit restricted cases (cf. Sec 4), the corollary carries over to MRMs with zero rewards. The duality theorem does not provide an algorithmic recipe for **CSRL**, but a direct solution using numerical integration can be constructed based on the fixpoint characterisation for $\mathcal{U}_J^I$. An investigation of the feasibility of applying known efficient performability evaluation algorithms to model checking **CSRL** is ongoing.

Typical performability measures. Performability measures that frequently appear in the literature, e.g., [15], can be specified by simple **CSRL**-formulas. This is illustrated by Table 1 where we listed a (non-exhaustive) variety of typical performability measures for the multi-processor system together with the corresponding **CSRL** formulas. Measure (a) expresses a bound on the steady-state availability of the system and (b) expresses (a bound on) the probability to be not in a failed state at time t, i.e., the instantaneous availability at time t. Measure (c) expresses the time until a failure, starting from a non-failed state. Evaluating this measure for varying t, gives us the distribution of the time to

Table 1. Performability measures and their logical specification

	performability measure	formula	logic
(a)	steady-state availability	$\mathcal{S}_{\bowtie p}(\neg F)$	**SL**
(b)	instantaneous availability at time t	$\mathcal{P}_{\bowtie p}(\Diamond^{[t,t]}\neg F)$	**CSL**
(c)	distribution of time to failure	$\mathcal{P}_{\bowtie p}(\neg F\mathcal{U}^{[0,t]}\, F)$	**CSL**
(d)	distribution of reward until failure	$\mathcal{P}_{\bowtie p}(\neg F\mathcal{U}_{[0,r]}\, F)$	**CRL**
(e)	distribution of cumulative reward until t	$\mathcal{P}_{\bowtie p}(\Diamond^{[t,t]}_{[0,r]}\mathrm{tt})$	**CSRL**

failure. Measure (d) complements this by expressing the distribution of the reward accumulated until failure. Measure (e) generalises (c) and (d) by expressing the simultaneous distribution of the accumulated reward against time, i.e., it expresses the probability for the reward accumulated at t to be at most r. This measure coincides with the performability distribution as proposed in the seminal paper [12]. Note that for the computation of all these measures efficient algorithms do exist [9]. We emphasize that, in its full generality, **CSRL** allows to specify much more complex performability measures than previous ad hoc methods.

A possible extension of CSRL. Consider state s in MRM $\mathcal{M}$. For time t and set of states S', the *instantaneous reward* $\rho^{\mathcal{M}}(s, S', t)$ equals $\sum_{s' \in S'} \pi^{\mathcal{M}}(s, s', t) \cdot \rho(s')$ and denotes the rate at which reward is earned in some state in S' at time t. The *expected (or long run) reward rate* $\rho^{\mathcal{M}}(s, S')$ equals $\sum_{s' \in S'} \pi^{\mathcal{M}}(s, s') \cdot \rho(s')$. We can now add the following operators to our framework:

$$
\begin{aligned}
s &\models \mathcal{E}_J(\Phi) \;\text{ iff } \rho^{\mathcal{M}}(s, Sat^{\mathcal{M}}(\Phi)) \in J \\
s &\models \mathcal{E}^t_J(\Phi) \;\text{ iff } \rho^{\mathcal{M}}(s, Sat^{\mathcal{M}}(\Phi), t) \in J \\
s &\models \mathcal{C}^I_J(\Phi) \;\text{ iff } \textstyle\int_I \rho^{\mathcal{M}}(s, Sat^{\mathcal{M}}(\Phi), u)\; du \in J
\end{aligned}
$$

Although the duality principle is not applicable to the new operators, their model checking is rather straightforward. The first two formulas require the summation of the Φ-conforming steady-state or transient state probabilities (as computed for measure (a) and (b)) multiplied with the corresponding rewards. The operator $\mathcal{C}^I_J(\Phi)$ states that the expected amount of reward accumulated in Φ-states during the interval I lies in J. It can be evaluated using a variant of uniformisation [9,16]. Some example properties are now: $\mathcal{E}_J(\neg F)$, which expresses the expected reward rate (e.g., the system's capacity) for an operational system, $\mathcal{E}^t_J(\mathrm{tt})$ expresses the expected instantaneous reward rate at time t and $\mathcal{C}^{[0,t]}_J(\mathrm{tt})$ expresses the amount of cumulated reward up to time t.

6 Concluding Remarks

We introduced a continuous-time, reward-based stochastic logic which is adequate for expressing performability measures of a large variety. Two important sub-logics were identified, viz. **CSL** [1,3], and the novel logic **CRL** that allows one to express reward-based properties. The main result of the paper is that **CSL**

and **CRL** are complementary, implying that **CRL**-properties for a Markov reward model can be interpreted as **CSL**-properties over a derived CTMC, so that existing model checking procedures for **CSL** can still be employed. The model checking of the full logic **CSRL**, in particular properties in which time- and reward-bounds are combined, is left for future work.

Acknowledgement. We thank the reviewers for their helpful comments.

References

1. A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Verifying continuous time Markov chains. In *CAV*, LNCS 1102, pp. 269–276, 1996.
2. C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *CAV*, LNCS, 2000.
3. C. Baier, J.-P. Katoen and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR*, LNCS 1664, pp. 146–162, 1999.
4. M.D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Trans. on Comp. Sys.*, **27**(6): 540–547, 1978.
5. G. Clark, S. Gilmore, and J. Hillston. Specifying performance measures for PEPA. In *Form. Meth. for Real-Time and Prob. Sys. (ARTS)*, LNCS 1601, pp. 211–227, 1999.
6. C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Found. of Comp. Sc. (FOCS)*, pp. 338–345, 1988.
7. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Form. Asp. of Comp.*, **6**: 512–535, 1994.
8. B.R. Haverkort and I.G. Niemegeers. Performability modelling tools and techniques. *Perf. Ev.*, **25**: 17–40, 1996.
9. B.R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach.* John Wiley & Sons, 1998.
10. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser and M. Siegle. A Markov chain model checker. In *TACAS*, LNCS 1785, pp. 347–362, 2000.
11. R.A. Howard. *Dynamic Probabilistic Systems; Vol. I, II.* John Wiley & Sons, 1971.
12. J.F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Trans. on Comp.*, **29**(8): 720–731, 1980.
13. J.F. Meyer. Performability evaluation: where it is and what lies ahead. In *1st IEEE Int. Comp. Perf. and Dependability Symp. (IPDS)*, pp. 334–343, 1995.
14. W.D. Obal and W.H. Sanders. State-space support for path-based reward variables. In *3rd IEEE Int. Comp. Perf. and Dependability Symp. (IPDS)*, pp. 228–237, 1998.
15. R.M. Smith, K.S. Trivedi and A.V. Ramesh. Performability analysis: measures, an algorithm and a case study. *IEEE Trans. on Comp.*, **37**(4): 406–417, 1988.
16. E. de Souza e Silva and H.R. Gail. Performability analysis of computer systems: from model specification to solution. *Perf. Ev.*, **14**: 157–196, 1992.
17. W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994.
18. K.S. Trivedi, J.K. Muppala, S.P. Woolet, and B.R. Haverkort. Composite performance and dependability analysis. *Perf. Ev.*, **14**: 197–215, 1992.
19. M.Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. *Found. of Comp. Sc. (FOCS)*, pp 327–338, 1985.

On the Representation of Timed Polyhedra*

Olivier Bournez[1] and Oded Maler[2]

[1] LORIA
Campus Scientifique BP 239
54506 Vandoeuvre lès Nancy, France
Olivier.Bournez@loria.fr

[2] VERIMAG
2, av. de Vignate
38610 Gières, France
maler@imag.fr

Abstract. In this paper we investigate *timed polyhedra*, i.e. polyhedra which are finite unions of full dimensional simplices of a special kind. Such polyhedra form the basis of timing analysis and in particular of verification tools based on timed automata. We define a representation scheme for these polyhedra based on their extreme vertices, and show that this compact representation scheme is canonical for all (*convex and non-convex*) polyhedra in *any* dimension. We then develop relatively efficient algorithms for membership, boolean operations, projection and passage of time for this representation.

1 Introduction and Motivation

Timed automata, automata augmented with clock variables [AD94], has proven to be a very useful formalism for modeling phenomena which involve both discrete transitions and quantitative timing information. Although their state-space is non-countable, the reachability problem, as well as other verification, synthesis and optimizations problems for timed automata are solvable. This is due to the fact that the clock space admits an equivalence relation (time-abstract bisimulation) of finite index, and it is hence sufficient to manipulate these equivalence classes, which form a restricted class of polyhedra that we call *timed polyhedra*.

Several verification tools for timed automata have been built during the last decade, e.g. Kronos [DOTY96,Y97], Timed Cospan [AK96] and Uppaal [LPY97], and the manipulation of timed polyhedra is the computational core of such tools. Difference bound matrices (DBM) are a well-known data-structure for representing *convex* timed polyhedra, but usually, in the course of verification, the accumulated reachable states can form a highly non-convex set whose representation and manipulation pose serious performance problems to these tools.

* This work was partially supported by the European Community Esprit-LTR Project 26270 VHS (Verification of Hybrid systems) and the French-Israeli collaboration project 970MAEFUT5 (Hybrid Models of Industrial Plants).

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 793–807, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Consequently, the search for new representation schemes for timed polyhedra is a very active domain of research [ABK+97], [BMPY97], [S99], [MLAH99b], [BLP+99], [W00].

In this paper, we propose a new representation scheme for non-convex timed polyhedra based on a reformulation and extension of our previous work in [BMP99] where we proposed a canonical representation for non-convex *orthogonal* polyhedra. As in [BMP99] the representation is based on a certain subset of the vertices of the polyhedron. The size of our canonical representation is $O(nd!)$ where n is the number of vertices and d is the dimension. Based on this representation we develop relatively-efficient algorithms for membership, Boolean operations, projection and passage of time on arbitrary timed polyhedra of any dimension. In order to simplify the presentation we restrict the discussion in this paper to *full-dimensional* timed polyhedra, but the results can be extended to treat unions of polyhedra of varying dimension.

The rest of the paper is organized as follows: in section 2 we define orthogonal polyhedra and give new proofs of the main results from [BMP99] concerning their representation by extreme vertices. In section 3 we introduce timed polyhedra and prove that they can be represented canonically by their extreme vertices. In section 4 we discuss Boolean operations, projections, and the calculation of the effect of time passage. Finally we mention some related work and future research directions.

2 Griddy Polyhedra and Their Representation

Throughout the paper we assume a d-dimensional $\mathbb{R}$-vector space. In this section we also assume a fixed basis for this space so that points and subsets can be identified through their coordinates by points and subset of $\mathbb{R}^d$. The results of this section are invariant under change of basis,[1] and this fact will be exploited in the next section.

We assume that all our polyhedra live inside a bounded subset $X = [0, m]^d \subseteq \mathbb{R}^d$ (in fact, the results hold also for $\mathbb{R}^d_+$). We denote elements of X as $\mathbf{x} = (x_1, \ldots, x_d)$, the zero vector by $\mathbf{0}$ and the vector $(1, 1, \ldots, 1)$ by $\mathbf{1}$. The *elementary grid* associated with X is $\mathbf{G} = \{0, 1 \ldots, m-1\}^d \subseteq \mathbb{N}^d$. For every point $\mathbf{x} \in X$, $\lfloor \mathbf{x} \rfloor$ is the grid point corresponding to the integer part of the components of $\mathbf{x}$. The grid admits a natural partial order defined as $(x_1, \ldots, x_d) \leq (x'_1, \ldots, x'_d)$ if for every i, $x_i \leq x'_i$. The set of subsets of the elementary grid forms a Boolean algebra $(2^{\mathbf{G}}, \cap, \cup, \sim)$ under the set-theoretic operations.

Definition 1 (Griddy Polyhedra). *Let $\mathbf{x} = (x_1, \ldots, x_d)$ be a grid point. The elementary box associated with $\mathbf{x}$ is the closed subset of X of the form $B(\mathbf{x}) = [x_1, x_1+1] \times [x_2, x_2+1] \times \ldots [x_d, x_d+1]$. The point $\mathbf{x}$ is called the leftmost corner of $B(\mathbf{x})$. The set of boxes is denoted by $\mathbf{B}$. A griddy polyhedron P is a union of elementary boxes, i.e. an element of $2^{\mathbf{B}}$.*

[1] Griddy polyhedra were called orthogonal polyhedra in [AA98,BMP99]. We prefer here the term griddy since the results do not depend on an orthogonal basis.

Although $2^{\mathbf{B}}$ is not closed under usual complementation and intersection, it is closed under the following operations:

$$A \sqcup B = A \cup B \qquad A \sqcap B = cl(int(A) \cap int(B)) \qquad \neg A = cl(\sim A)$$

(where cl and int are the topological closure and interior operations[2]). The bijection B between $\mathbf{G}$ and $\mathbf{B}$ which associates every box with its leftmost corner clearly induces an isomorphism between $(2^{\mathbf{G}}, \cap, \cup, \sim)$ and $(2^{\mathbf{B}}, \sqcap, \sqcup, \neg)$. In the sequel we will switch between point-based and box-based terminology according to what serves better the intuition.

Definition 2 (Color Function). *Let P be an griddy polyhedron. The color function $c : X \to \{0, 1\}$ is defined as follows: if $\mathbf{x}$ is a grid point then $c(\mathbf{x}) = 1$ iff $B(\mathbf{x}) \subseteq P$; otherwise, $c(\mathbf{x}) = c(\lfloor \mathbf{x} \rfloor)$.*

Note that c almost coincides with the characteristic function of P as a subset of X. It differs from it only on right-boundary points (see Figure 1-(a)).

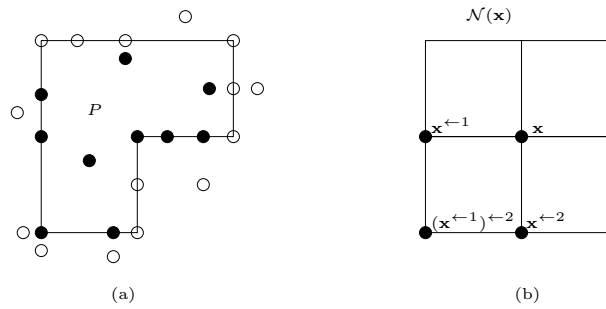


Fig. 1. (a) A griddy polyhedron and a sample of the values of the color function it induces on X. (b) The neighborhood and predecessors of a grid point $\mathbf{x}$.

Definition 3 (Predecessors, Neighborhoods and Cones). *In the following we consider $\mathbf{x}$ to be a grid point $\mathbf{x} = (x_1, \ldots, x_d)$.*

- *The i-predecessor of $\mathbf{x}$ is $\mathbf{x}^{\leftarrow i} = (x_1, \ldots, x_i - 1, \ldots, x_d)$.* We use $\mathbf{x}^{\leftarrow ij}$ as a shorthand for $(\mathbf{x}^{\leftarrow i})^{\leftarrow j}$.
- *The neighborhood of $\mathbf{x}$ is the set $\mathcal{N}(\mathbf{x}) = \{x_1 - 1, x_1\} \times \ldots \times \ldots \{x_d - 1, x_d\}$,* i.e. the vertices of a box lying between $\mathbf{x} - \mathbf{1}$ and $\mathbf{x}$, (Figure 1-(b)).
- *The backward cone based at $\mathbf{x}$ is ${}_{\triangleright}\mathbf{x} = \{\mathbf{y} \in \mathbf{G} : \mathbf{y} \leq \mathbf{x}\}$* (Figure 2-(a)).
- *The forward cone based at $\mathbf{x}$ is $\mathbf{x}^{\triangleleft} = \{\mathbf{y} \in \mathbf{G} : \mathbf{x} \leq \mathbf{y}\}$* (Figure 2-(b)).

[2] See [B83] for definitions.

Every grid point $\mathbf{x}$ is contained in all the forward cones $\mathbf{y}^{\triangleleft}$ such that $\mathbf{y} \in {}_{\triangleright}\mathbf{x}$.

Let $\oplus$ denote addition modulo 2, known also as the exclusive-or (XOR) operation in Boolean algebra, $p \oplus q = (p \wedge \neg q) \vee (\neg p \wedge q)$. We will use the same notation for the symmetric set difference operation on sets, defined as $A \oplus B = \{\mathbf{x} : (\mathbf{x} \in A) \oplus (\mathbf{x} \in B)\}$.[3] This is an associative and commutative operation, satisfying $A \oplus A = 0$ on numbers, and $A \oplus A = \emptyset$ on sets. We will show that every griddy polyhedron admits a canonical decomposition into a XOR of cones.

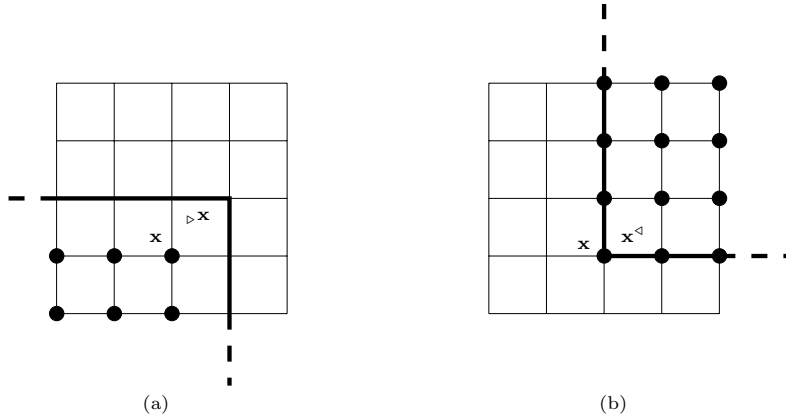


Fig. 2. (a) The backward cone based at $\mathbf{x}$. (b) The forward cone based at $\mathbf{x}$.

Theorem 1 ($\oplus$-representation). *For any griddy polyhedron P there is a unique finite set V of grid points such that $P = \bigoplus_{\mathbf{v} \in V} \mathbf{v}^{\triangleleft}$.*

Proof. First we show existence of V. Let $\prec$ be a total order relation on $\mathbf{G}$, $\mathbf{x}_1 \prec \mathbf{x}_2 \prec \ldots \mathbf{x}_{m^d}$, which is consistent with the partial order $\leq$ (any lexicographic or diagonal order will do). By definition, $\mathbf{x} \prec \mathbf{y}$ implies $\mathbf{y} \notin {}_{\triangleright}\mathbf{x}$. The following iterative algorithm constructs V. We denote by V_j the value of V after j steps and by c_j the color function of the associated polyhedron $P_j = \bigoplus_{\mathbf{v} \in V_j} \mathbf{v}^{\triangleleft}$.

```
V_0:=∅
for j = 1 to m^d do
  if c(x_j) ≠ c_{j-1}(x_j)
    then V:=V ∪ {x_j}
end
```

This algorithm is correct because at the end of every step j in the loop we have $c_j(\mathbf{x}_k) = c(\mathbf{x}_k)$ for every $k \leq j$. It holds trivially at the beginning and in

[3] Note that on $2^{\mathbf{B}}$, $A \oplus B$ should be modified into $cl(int(A) \oplus int(B))$.

every step, the color of $\mathbf{x}_k$ for all $k < j$ is preserved (because their color is not influenced by the cone $\mathbf{x}_j^{\triangleleft}$) and updated correctly for $\mathbf{x}_j$ if needed.

For uniqueness we show that if there are two sets of points U and V such that

$$P = \bigoplus_{\mathbf{v} \in V} \mathbf{v}^{\triangleleft} = \bigoplus_{\mathbf{u} \in U} \mathbf{u}^{\triangleleft}$$

then $U = V$. Indeed, by the properties of $\oplus$, we have

$$\emptyset = P \oplus P = \bigoplus_{\mathbf{v} \in V} \mathbf{v}^{\triangleleft} \oplus \bigoplus_{\mathbf{u} \in U} \mathbf{u}^{\triangleleft} = \bigoplus_{\mathbf{y} \in U \oplus V} \mathbf{y}^{\triangleleft}$$

(if a point $\mathbf{v}$ appears in both U and V it can be eliminated because $\mathbf{v}^{\triangleleft} \oplus \mathbf{v}^{\triangleleft} = \emptyset$). The set denoted by the rightmost expression is empty only if $U \oplus V = \emptyset$, otherwise any minimal vertex $\mathbf{v}$ in $U \oplus V$ belongs to it. □

Remark: Since every set of points defines a $\oplus$-formula and a polyhedron, we have an interesting non-trivial bijection on subsets of $\mathbf{G}$. Note that for the chess board $V = \mathbf{G}$.

Let $\mathcal{V} : \mathbf{G} \to \{0, 1\}$ be the characteristic function of the set $V \subseteq \mathbf{G}$.

Observation 1. *For every point* $\mathbf{x}$, $c(\mathbf{x}) = \bigoplus_{\mathbf{y} \in {}_{\triangleright}\mathbf{x}} \mathcal{V}(\mathbf{y})$

This gives us immediately a decision procedure for the membership problem $\mathbf{x} \in P$: just check the parity of $V \cap {}_{\triangleright}\mathbf{x}$. In [BMP99] we have shown that the elements of V are those vertices of P that satisfy some local conditions. In the following we give an alternative proof of this fact, which requires some additional definitions to facilitate induction over the dimensions.

Definition 4 (k-Neighborhood and k-Backward Cone). *Let* $\mathbf{x} = (x_1, \ldots, x_d)$.

- *The k-neighborhood of* $\mathbf{x}$ *is the set*

$$N^k(\mathbf{x}) = \{x_1 - 1, x_1\} \times \ldots \times \{x_k - 1, x_k\} \times \{x_{k+1}\} \times \ldots \times \{x_d\}.$$

- *The k-backward cone of* $\mathbf{x}$ *is the set*

$$M^k(\mathbf{x}) = \{x_1\} \times \ldots \times \{x_k\} \times \{0, \ldots, x_{k+1}\} \times \ldots \times \{0, \ldots, x_d\}.$$

Note that $N^0(\mathbf{x}) = \{\mathbf{x}\}$, $N^d(\mathbf{x}) = \mathcal{N}(\mathbf{x})$, $M^0(\mathbf{x}) = {}_{\triangleright}\mathbf{x}$ and $M^d(\mathbf{x}) = \{\mathbf{x}\}$ (see Figure 3).

Observation 2. *For every* $k \geq 0$,

$$N^k(\mathbf{x}) = N^{k-1}(\mathbf{x}) \oplus N^{k-1}(\mathbf{x}^{\leftarrow k}) \quad \textit{and} \quad M^k(\mathbf{x}) = M^{k-1}(\mathbf{x}) \oplus M^{k-1}(\mathbf{x}^{\leftarrow k}).$$

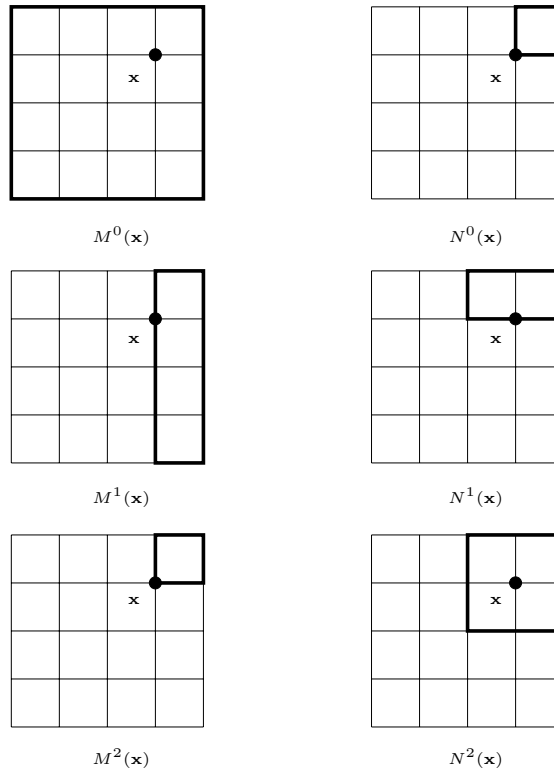


Fig. 3. The k-backward cone $M^k(\mathbf{x})$ and the k-neighborhood $N^k(\mathbf{x})$ for $k = 0, 1, 2$.

Theorem 2 (Extreme Vertices). *A point* $\mathbf{x}$ *is a basis for a cone in the canonical decomposition of a griddy polyhedron* P *if and only if* $\mathbf{x}$ *is a vertex of* P *satisfying*

$$\bigoplus_{\mathbf{y} \in \mathcal{N}(\mathbf{x})} c(\mathbf{y}) = 1 \tag{1}$$

Proof. First we prove that for every $\mathbf{x} \in \mathbf{G}$ and every k, $0 \leq k \leq d$

$$\bigoplus_{\mathbf{y} \in N^k(\mathbf{x})} c(\mathbf{y}) = \bigoplus_{\mathbf{y} \in M^k(\mathbf{x})} \mathcal{V}(\mathbf{y}). \tag{2}$$

The proof is done by induction on k. For $k = 0$ it is just a rephrasing of observation 1. Suppose it holds for $k - 1$. By summing up the claims for $\mathbf{x}$ and for $\mathbf{x}^{\leftarrow k}$ we obtain

$$\bigoplus_{\mathbf{y} \in N^{k-1}(\mathbf{x})} c(\mathbf{y}) \oplus \bigoplus_{\mathbf{y} \in N^{k-1}(\mathbf{x}^{\leftarrow k})} c(\mathbf{y}) = \bigoplus_{\mathbf{y} \in M^{k-1}(\mathbf{x})} \mathcal{V}(\mathbf{y}) \oplus \bigoplus_{\mathbf{y} \in M^{k-1}(\mathbf{x}^{\leftarrow k})} \mathcal{V}(\mathbf{y})$$

which, by virtue of Observation 2 and the inductive hypothesis, gives the result for k. Substituting $k = d$ in (2), we characterize the elements of V as those

satisfying condition (1). In [BMP99] we have proved that these points constitute a subset of the vertices of P which we call "extreme" vertices, following a geometrical definition in [AA98] for $d \leq 3$. □

We review our main algorithmic results from [BMP99], assuming a fixed dimension d and denoting by n_P the number of extreme vertices of a polyhedron P, which we assume to be sorted in some fixed order.

Observation 3 (Boolean Operations). *Let A and B be two griddy polyhedra.*

- *The symmetric difference of A and B, $A \oplus B$, can be computed in time $O(n_A + n_B)$.*
- *The complement of A, $X - A$, can be obtained in time $O(1)$.*
- *The union, $A \cup B$, and the intersection, $A \cap B$, can be computed in time $O(n_A n_b)$.*

Proof. Computing $A \oplus B$ is trivial and so is complementation using $X - A = \mathbf{0}^{\triangleleft} \oplus A$. Observing that $A \cup B = A \oplus B \oplus A \cap B$, computing union reduces to computing intersection. Now, using recursively the identity $(A \oplus B) \cap C = (A \cap C) \oplus (B \cap C)$, write

$$(\bigoplus_{\mathbf{x}_i} \mathbf{x}_i^{\triangleleft}) \cap (\bigoplus_{\mathbf{y}_j} \mathbf{y}_j{}^{\triangleleft}) = \bigoplus_{\mathbf{x}_i, \mathbf{y}_j} (\mathbf{x}_i{}^{\triangleleft} \cap \mathbf{y}_j{}^{\triangleleft}) = \bigoplus_{\mathbf{x}_i, \mathbf{y}_j} \max(\mathbf{x}_i, \mathbf{y}_j)^{\triangleleft}$$

where max denotes the maximum component-wise. □

We use the following terminology from [BMP99]:

Definition 5 (Slices, Sections and Cones). *Let P be a griddy polyhedron, $i \in \{1, \ldots, d\}$ and $z \in \{0, \ldots, m-1\}$.*

- *The (i, z)-slice of P is the d-dimensional polyhedron $J_{i,z}(P) = P \sqcap \{\mathbf{x} : z \leq x_i \leq z + 1\}$.*
- *The (i, z)-section of P is the $(d-1)$-dimensional polyhedron $\mathcal{J}_{i,z}(P) = J_{i,z}(P) \cap \{\mathbf{x} : x_i = z\}$.*
- *The i-projection of P is the $(d-1)$-dimensional polyhedron $\pi_{\downarrow i}(P) = \{(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_d) : \exists z \ (x_1, \ldots, x_{i-1}, z, x_{i+1}, \ldots x_d) \in P\}$.*

Observation 4 (Computation of Slice, Section and Projection). *The computation of $J_{i,z}$ and of $\mathcal{J}_{i,z}$ can be performed in time $O(n)$. The projection $\pi_{\downarrow i}(P)$ of P is computable in time $O(n^3)$.*

Proof. Using identity $(A \oplus B) \sqcap C = (A \sqcap C) \oplus (B \sqcap C)$, we have

$$J_{i,z}(\bigoplus_{\mathbf{x} \in V} \mathbf{x}^{\triangleleft}) = \bigoplus_{\mathbf{x} \in V} (J_{i,z}(\mathbf{x}^{\triangleleft})).$$

This gives an immediate algorithm for computing $J_{i,z}(P)$ and, hence for computing $\mathcal{J}_{i,z}(P)$. For projection write $\pi_{\downarrow i}(P) = \bigsqcup_z \pi_{\downarrow i}(J_{i,z}(P))$. □

3 Timed Polyhedra and Their Representation

In this section we extend our results to *timed polyhedra* whose building blocks are certain types of simplices. Let Π denote the set of permutations on $\{1, \ldots, d\}$. We write permutations $\sigma \in \Pi$ as $(\sigma(1)\ \sigma(2)\ \cdots\ \sigma(d))$. There is a natural bijection between Π and the set of simplices occurring in a specific triangulation of the d-unit hypercube (sometimes called the Kuhn triangulation [L97]). Hence the correspondence between grid points and elementary boxes from the previous section can be extended into a correspondence between $\mathbf{G} \times \Pi$ and the set of such triangulations of the elementary boxes in X.

Definition 6 (Elementary Simplices, Timed Polyhedra and Cones). *Let* $\mathbf{v}$ *be a grid point and let* σ *be a permutation.*

- *The elementary simplex associated with* $(\mathbf{v}, \sigma)$ *is*

$$B(\mathbf{v}, \sigma) = \{\mathbf{x} : 0 \leq x_{\sigma(1)} - v_{\sigma(1)} \leq x_{\sigma(2)} - v_{\sigma(2)} \leq \ldots \leq x_{\sigma(d)} - v_{\sigma(d)} \leq 1\}.$$

- *A timed polyhedron is a union of elementary simplices. It can be expressed as a color function* $c : \mathbf{G} \times \Pi \to \{0, 1\}$.
- *The* σ*-forward cone based on a point* $\mathbf{v}$ *is*

$$(\mathbf{v}, \sigma)^{\triangleleft} = \{\mathbf{x} : 0 \leq x_{\sigma(1)} - v_{\sigma(1)} \leq x_{\sigma(2)} - v_{\sigma(2)} \leq \ldots \leq x_{\sigma(d)} - v_{\sigma(d)}\}.$$

These notions are illustrated in Figure 4 for $d = 2$. In the sequel we will use the word simplex both for a pair $(\mathbf{v}, \sigma)$ and for the set $B(\mathbf{v}, \sigma)$. Note that for every $\mathbf{v}$,

$$\bigoplus_{\sigma \in \Pi} B(\mathbf{v}, \sigma) = B(\mathbf{v}) \quad \text{and} \quad \bigoplus_{\sigma \in \Pi} (\mathbf{v}, \sigma)^{\triangleleft} = \mathbf{v}^{\triangleleft}.$$

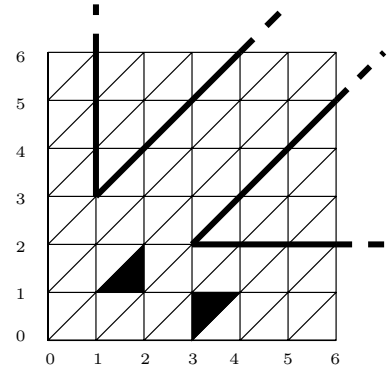


Fig. 4. The set of basic simplices in dimension 2, the simplices $B((1,1),(2\ 1))$ and $B((3,0),(1\ 2))$ and the forward-cones $((3,2),(2\ 1))^{\triangleleft}$ and $((1,3),(1\ 2))^{\triangleleft}$.

Theorem 3 ($\oplus$-Representation of Timed Polyhedra). *For any timed polyhedron P, there is a unique finite set $V \subseteq \mathbf{G} \times \Pi$ such that*

$$P = \bigoplus_{(\mathbf{v},\sigma)\in V} (\mathbf{v},\sigma)^{\triangleleft}$$

Proof. The proof is similar to the proof of Theorem 1. The only additional detail is the extension of the order on $\mathbf{G}$ into a lexicographic order on $\mathbf{G} \times \Pi$. Since the cones ${}_{\triangleright}(\mathbf{x},\sigma)$ and ${}_{\triangleright}(\mathbf{x},\tau)$ are disjoint for $\sigma \neq \tau$, the properties used in the proof are preserved. □

Our goal for the rest of the section is to find a *local characterization* of the elements $(\mathbf{v},\sigma)$ of V, similar to property (1) in Theorem 2, based on the parity of the colors on some suitably-defined *neighborhood* of $(\mathbf{v},\sigma)$.

Observation 5 (Decomposition of Timed Polyhedra). *Any timed polyhedron P can be decomposed into*

$$P = \bigoplus_{\sigma\in\Pi} P_\sigma.$$

where each P_σ is a griddy polyhedron in some basis of $\mathbb{R}^d$.

Proof. By letting $V_\sigma = \{\mathbf{v} : (\mathbf{v},\sigma) \in V\}$ we can rewrite P as

$$P = \bigoplus_{\sigma\in\Pi} \bigoplus_{\mathbf{v}\in V_\sigma} (\mathbf{v},\sigma)^{\triangleleft} = \bigoplus_{\sigma\in\Pi} P_\sigma.$$

Each P_σ is a XOR of σ-cones, and hence it is griddy in the coordinate system corresponding to σ which is related to the orthogonal system by the transformation $y_{\sigma(1)} = x_{\sigma(1)}$ and $y_{\sigma(i)} = x_{\sigma(i)} - x_{\sigma(i-1)}$ for $i \geq 2$. □

We denote the color function of P_σ by c_σ. We call the corresponding grid the *σ-grid*. In this grid the i-predecessor of $\mathbf{x}$ is denoted by $\mathbf{x}^{\overset{\sigma}{\leftarrow} i}$, and the neighborhood of $\mathbf{x}$ by $N_\sigma(\mathbf{x})$ (see Figure 5). By definition $(\mathbf{x},\sigma) \in V$ iff if $\mathbf{x} \in V_\sigma$, and, since Theorem 2 does not depend on orthogonality, we have:

Observation 6. *A simplex $(\mathbf{x},\sigma)$ occurs in V iff* $\bigoplus_{\mathbf{y}\in N_\sigma(\mathbf{x})} c_\sigma(\mathbf{y},\sigma) = 1.$

This definition is based on P_σ, not on P itself. We will show that this sum can be reduced to the sum of the values of c on a certain set of simplices $S(\mathbf{x},\sigma)$, to be defined below.

Definition 7 (Permutation and Simplex Predecessors). *With every $i \in \{1,\ldots,d\}$ define the i-predecessor function ${}^{\leftarrow i} : \Pi \to \Pi$ such that for every σ such that $\sigma(k) = i$*

$$\sigma^{\leftarrow i}(j) = \begin{cases} \sigma(j) & \text{if } j < k \\ \sigma(j+1) & \text{if } k < j < d \\ i & \text{if } j = d \end{cases}$$

The i-predecessor of a simplex $(\mathbf{x},\sigma)$, for $i \in \{1,\ldots,d\}$ is defined as $(\mathbf{x},\sigma)^{\leftarrow i} = (\mathbf{x}^{\leftarrow i}, \sigma^{\leftarrow i})$.

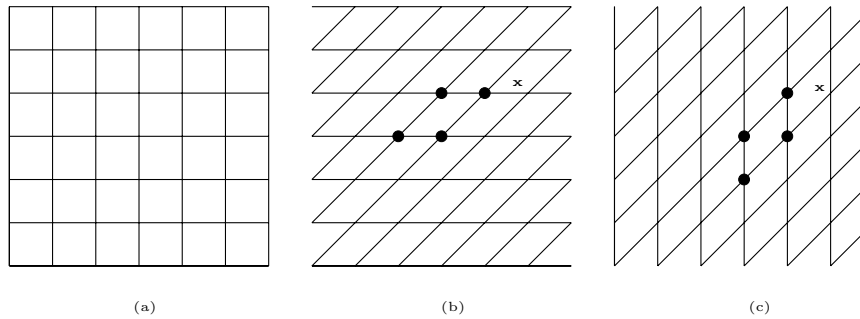


Fig. 5. (a) The orthogonal grid. (b) The (2 1)-grid and $N_{(2\ 1)}(\mathbf{x})$. (c) The (1 2)-grid and $N_{(1\ 2)}(\mathbf{x})$.

In other words, $\sigma^{\leftarrow i}$ is the permutation obtained from σ by picking x_i and putting it at the end of the ordering. We denote by $\sigma^{\leftarrow ij}$ the successive application of the operator $(\sigma^{\leftarrow i})^{\leftarrow j}$. Note that unlike the analogue definition on the grid, the permutation predecessors is not commutative, i.e. $\sigma^{\leftarrow ij} \neq \sigma^{\leftarrow ji}$.

The fact that $(\mathbf{y}, \tau) = (\mathbf{x}, \sigma)^{\leftarrow i}$ has the following geometrical interpretation: $B(\mathbf{y}, \tau)$ is the first simplex outside $B(\mathbf{x})$ encountered while going backward in direction i from $B(\mathbf{x}, \sigma)$. We can lift these functions into ${}^{\leftarrow i} : 2^{\mathbf{G}\times\Pi} \to 2^{\mathbf{G}\times\Pi}$ in the natural way. The following definition is crucial for this paper. It specifies which neighborhood of a simplex determines its membership in V.

Definition 8 (Simplex k-Neighborhood). *The simplex k-neighborhood of a simplex $(\mathbf{x}, \sigma)$ is the set of simplices defined recursively as: $S^0(\mathbf{x}, \sigma) = \{(\mathbf{x}, \sigma)\}$ and*

$$S^{k+1}(\mathbf{x}, \sigma) = S^k(\mathbf{x}, \sigma) \cup (S^k(\mathbf{x}, \sigma))^{\leftarrow \sigma(d-k)}.$$

This notion is depicted in Figure 6. We write S for S^d. Note that unlike the definition of neighborhood for griddy polyhedra (Definition 4), where the recursion over dimensions can be done in any order, here the order is important and depends on the permutation.
We intend to show that

$$\bigoplus_{\mathbf{y}\in N_\sigma(\mathbf{x})} c_\sigma(\mathbf{y}, \sigma) = \bigoplus_{(\mathbf{y},\tau)\in S(\mathbf{x},\sigma)} c(\mathbf{y}, \tau).$$

Definition 9 (Close Permutations). *With every $\sigma \in \Pi$ and every i, $0 \leq i \leq d$, we associate a set of permutations defined recursively as $\Pi_\sigma^0 = \Pi$ and*

$$\Pi_\sigma^{i+1} = \Pi_\sigma^i \cap \{\tau : \sigma(i) = \tau(i)\}.$$

In other words, Π_σ^i is the set of permutation that agree with σ on the first i coordinates.

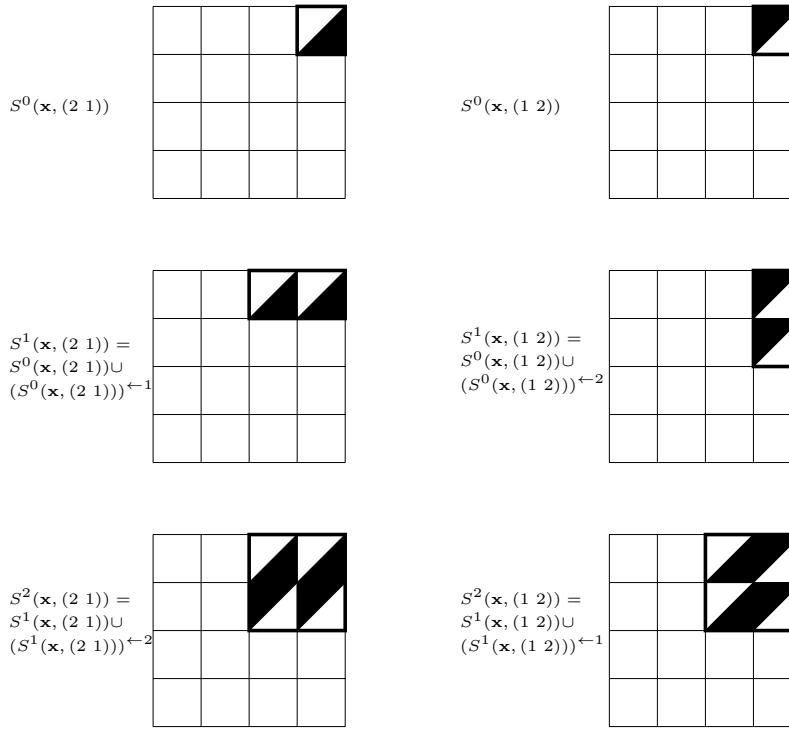


Fig. 6. The simplex k-neighborhoods of two simplices.

Note, of course, that $\Pi_\sigma^d = \{\sigma\}$. With every such set of permutations we associate the polyhedron $P_\sigma^i = \bigoplus_{\tau \in \Pi_\sigma^i} P_\tau$ and let c_σ^i denote its corresponding color function. For every k, $0 \leq k \leq d$, let $\rho_k(\mathbf{x}, \sigma)$ denote the quantity

$$\rho_k(\mathbf{x}, \sigma) = \bigoplus_{s \in S^k(\mathbf{x}, \sigma)} c_\sigma^{d-k}(s)$$

Observation 7 (Fundamental Property). *Let $(\mathbf{v}, \sigma)$ be a simplex, and let τ be a permutation in Π_σ^{i-1} for some $i \in \{i, \ldots, d\}$. Then:*

- $c_\tau((\mathbf{v}, \sigma)^{\leftarrow \sigma(i)}) = c_\tau(\mathbf{v}, \sigma)$ *when* $\tau \notin \Pi_\sigma^i$.
- $c_\tau((\mathbf{v}, \sigma)^{\leftarrow \sigma(i)}) = c_\tau(\mathbf{v}^{\overset{\sigma}{\leftarrow} \sigma(i)}, \sigma)$ *when* $\tau \in \Pi_\sigma^i$.

Proof. Working in the coordinate system on which P_σ is griddy, i.e. $y_{\sigma(1)} = x_{\sigma(1)}$ and $y_{\sigma(i)} = x_{\sigma(i)} - x_{\sigma(i-1)}$ for $i \geq 2$, it is easy to see that $(\mathbf{v}, \sigma)$ and $(\mathbf{v}, \sigma)^{\leftarrow \sigma(i)}$ are both included in a same elementary box when $\tau \notin \Pi_\sigma^i$, and are included in two different consecutive boxes in direction $\sigma(i)$ when $\tau \in \Pi_\sigma^i$. $\square$

Observing that any simplex $(\mathbf{v}, \tau) \in S^k(\mathbf{x}, \sigma)$ satifies $\tau \in \Pi_\sigma^{d-k}$, we obtain, for every k:

$$\rho_{k+1}(\mathbf{x}, \sigma) = \rho_k(\mathbf{x}, \sigma) \oplus \rho_k(\mathbf{x}^{\overset{\sigma}{\leftarrow}\sigma(d-k)}, \sigma) \tag{3}$$

Theorem 4 (Main Result). *A cone $(\mathbf{x}, \sigma)^\triangleleft$ occurs in the canonical decomposition of a timed polyhedron P iff $\mathbf{x}$ is a vertex of P satisfying condition*

$$\bigoplus_{(\mathbf{y},\tau)\in S^d(\mathbf{x},\sigma)} c(\mathbf{y}, \tau) = 1 \tag{4}$$

Proof. From (3) we have that $\rho_d(\mathbf{x}, \sigma)$ corresponds to $\bigoplus_{\mathbf{y}\in N_\sigma(\mathbf{x})} c_\sigma(\mathbf{y}, \sigma)$, and hence it indicates the extremity of $(\mathbf{x}, \sigma)$.

It remains to prove that an extreme point is a vertex by induction over d. Case $d = 1$ is immediate by a systematic inspection. Now, for $d \geq 1$, $S(\mathbf{x}, \sigma)$ is the union of $S^+ = S^{d-1}(\mathbf{x}, \sigma)$ and $S^- = (S^{d-1}(\mathbf{x}, \sigma))^{\leftarrow\sigma(1)}$. Observing that these two sets are separated by hyper-plane H of equation $x_{\sigma(1)} = \mathbf{x}_{\sigma(1)}$ and that the sum of c on S^+ and S^- must differ, $\mathbf{x}$ must belong to a facet included in H. Interchanging, if necessary, S^+ and S^-, assume that sum of c on S^+ is 1 (and 0 on S^-). Pushing away coordinate $x_{\sigma(1)}$, the sum of c on S^+ reduces to the extremity condition in dimension $d - 1$. The induction hypothesis says that $\mathbf{x}$ must belong to some edge linearly independent of H. As a consequence $\mathbf{x}$ must be a vertex. □

4 Algorithms on Timed Polyhedra

In this section we describe operations on timed polyhedra using the extreme vertices representation, assuming dimension d fixed, and using n_P to denote represent the number of $(\mathbf{x}, \sigma)$ pairs in the canonical decomposition of P, stored in some fixed order.

Theorem 5 (Boolean operations). *Complementation, symmetric difference and union (or intersection) of timed polyhedra A and B can be computed in time $O(1)$, $O(n_A + n_B)$ and $O(n_A n_B)$ respectively.*

Proof. Exactly as in Theorem 3. □

The two other operations needed for the verification of timed automata are the projection of a timed polyhedron on $x_i = 0$, which corresponds to resetting a clock variable to zero, and the forward time cone which corresponds to the effect of time passage which increases all clock variables uniformly. As in griddy polyhedra we will use the slicing operation for the computation.

Definition 10 (Slice, Projection and Forward Time Cone). *Let P be a timed polyhedron, z an integer in $[0, m)$ and σ a permutation.*

- *The* $(i,z)-\sigma$*-slice of* P *is the* d*-dimensional polyhedron* $J^{\sigma}_{i,z}(P) = P \sqcap \bigsqcup_{\{\mathbf{x}:x_i=z\}} B(\mathbf{x},\sigma)$.
- *The* i*-projection of* P *is the* $(d-1)$*-dimensional polyhedron* $\pi_{\downarrow i}(P) = \{(x_1,\ldots,x_{i-1},x_{i+1},\ldots x_d) : \exists z\ (x_1,\ldots,x_{i-1},z,x_{i+1},\ldots x_d) \in P\}$.
- *The forward time cone of* P *is the* d*-dimensional polyhedron* $P^{\Rightarrow} = \{\mathbf{x} : \exists t \geq 0\ \mathbf{x} - t\mathbf{1} \in P\}$.

These operations are illustrated in Figure 7.

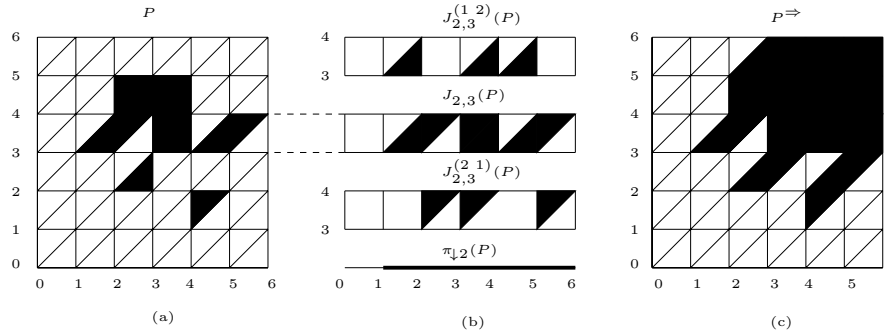


Fig. 7. A timed polyhedron P (a), some of its slices and projections (b) and its forward time cone (c).

Theorem 6 (Computation of Slice, Projection and Time Cone). *Given a timed polyhedron* P*,*

- *The computation of of* $J^{\sigma}_{i,z}(P)$ *can be done in time* $O(n_P)$.
- *The computation of* $\pi_{\downarrow i}(P)$ *can be done is time* $O(n_P^3)$.
- *The computation of* $P^{\Rightarrow}$ *can be done is time* $O(n_P^3)$.

Sketch of proof: The definition of $J^{\sigma}_{i,z}(P)$ in terms of vertex-permutation pairs is $J^{\sigma}_{i,z}(P) = P \cap \{(\mathbf{x},\sigma) : x_i = z\}$. Hence the result follows immediately from the identity $(A \oplus B) \sqcap C = (A \sqcap C) \oplus (B \sqcap C)$ which reduces the problem into n_P intersections of cones with a hyper-plane. For every i

$$P = \bigcup_{z\in[0,m]} \bigcup_{\sigma\in\Pi} J^{\sigma}_{i,z}(P),$$

hence, since both projection and time cone distribute over union they can be written as

$$\pi_{\downarrow i}(P) = \bigcup_{z\in[0,m]} \bigcup_{\sigma\in\Pi} \pi_{\downarrow i}(J^{\sigma}_{i,z}(P))$$

and

$$P^{\Rightarrow} = \bigcup_{z\in[0,m]} \bigcup_{\sigma\in\Pi} (J^{\sigma}_{i,z}(P))^{\Rightarrow}.$$

Projection of a slices is trivial and time cone of a slice is obtained by replacing every elementary simplex by a forward cone. □

5 Past, and Future Work

In this paper, we introduced a new canonical representation scheme for timed polyhedra as well as algorithms for performing the operations required for reachability analysis of timed automata. To the best of our knowledge these results are original. The representation of polyhedra is a well-studied problem, but most of the computational geometry and solid modeling literature is concerned only with low dimensional polyhedra (motivated by computer graphics) or with convex polyhedra for which a very nice theory exists. No such theory exist for non-convex polyhedra (see, for example, [A98] and the references therein).

The closest work to ours was that of [AA97,AA98], which we strengthened and generalized to arbitrary dimension in [BMP99] and extended from orthogonal to timed polyhedra in the present paper. The fact that non-convex polyhedra of arbitrary dimension can be represented using a $\oplus$-formula is not new (see for example a recent result in [E95]) but so far only for griddy and timed polyhedra a natural canonical form has been found.

We intend to implement this representation scheme and its corresponding algorithms, as we did for griddy polyhedra, and to see how the performance compares with other existing methods. Although the reachability problem for timed automata is intractable, practically, the manipulated polyhedra might turn out to have few vertices. In addition to the potential practical value we believe that timed polyhedra are interesting mathematical objects whose study leads to a nice theory.

References

A98. A. Aguilera, Orthogonal Polyhedra: Study and Application, PhD Thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 1998.

AA97. A. Aguilera and D. Ayala, Orthogonal Polyhedra as Geometric Bounds, in Constructive Solid Geometry, *Proc. Solid Modeling 97*, 1997

AA98. A. Aguilera and D. Ayala, Domain Extension for the Extreme Vertices Model (EVM) and Set-membership Classification, *Proc. Constructive Solid Geometry 98*, 1998.

AD94. R. Alur and D.L. Dill, A Theory of Timed Automata, *Theoretical Computer Science* 126, 183–235, 1994.

A98. R. Alur, Timed Automata, in *NATO-ASI Summer School on Verification of Digital and Hybrid Systems*, 1998 and in *11th International Conference on Computer-Aided Verification*, LNCS 1633, 8-22, Springer, 1999.

AK96. R. Alur, and R.P. Kurshan, Timing Analysis in COSPAN, in R. Alur, T.A. Henzinger and E. Sontag (Eds.), *Hybrid Systems III*, LNCS 1066, 220-231, Springer, 1996.

ABK+97. E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli and A. Rasse, Data-Structures for the Verification of Timed Automata, in O. Maler (Ed.), *Proc. HART'97*, LNCS 1201, 346-360, Springer, 1997.

BLP+99. G. Behrmann, K. Larsen, J. Pearson, C. Weise and W. Yi, Efficient Timed Reachability Analysis Using Clock Difference Diagrams, *Proc. CAV'99*, Springer, 1999.

BMP99. O. Bournez, O. Maler and A. Pnueli, Orthogonal Polyhedra: Representation and Computation, in F. Vaandrager and J. van Schuppen (Eds.), *Hybrid Systems: Computation and Control*, LNCS 1569, 46-60, Springer, 1999.

BMPY97. M. Bozga, O. Maler, A. Pnueli, S. Yovine, Some Progress in the Symbolic Verification of Timed Automata, in O. Grumberg (Ed.) *Proc. CAV'97*, 179-190, LNCS 1254, Springer, 1997.

B83. A. Brondsted, *An Introduction to Convex Polytopes*, Springer, 1983.

DOTY96. C. Daws, A. Olivero, S. Tripakis, and S. Yovine, The Tool KRONOS, in R. Alur, T.A. Henzinger and E. Sontag (Eds.), *Hybrid Systems III*, LNCS 1066, 208-219, Springer, 1996.

E95. H. Edelsbrunner, Algebraic decompositions of non-convex polyhedra, in *Proc. 36th Annual Symposium on Foundations of Computer Science*, 248-257, Milwaukee, Wisconsin, 1995.

LPY97. K.G. Larsen, P. Pettersson and W. Yi, UPPAAL in a Nutshell, *Software Tools for Technology Transfer* 1/2, 1997.

L97. C.W. Lee, Subdivision and Triangulation of Polytopes, in J.E Goodman and J. O'Rourke (Eds.), *Handbook of Discrete and Computational Geometry*, CRC Press, 1997.

MLAH99b. J. Møller, J. Lichtenberg, H.R. Andersen, and H. Hulgaard, Difference Decision Diagrams, *Proc. CSL'99*, 1999.

S99. K. Strehl, Interval Diagrams: Increasing Efficiency of Symbolic Real-Time Verification, *Proc. RTCSA '99*, 188-491, 1999.

Y97. S. Yovine, Kronos: A verification tool for real-time systems *Software Tools for Technology Transfer* 1/2, 1997.

W00. F. Wang, Efficient Data Structure for Fully Symbolic Verification of Real-Time Software Systems, in S. Graf and M. Schwartzbach (Eds.), *Proc. TACAS'00*, 157-151, LNCS 1785, Springer, 2000.

Min-wise Independent Permutations: Theory and Practice

Andrei Z. Broder

AltaVista Company
1825 S. Grant Street, Suite 200
San Mateo, CA 94402, USA
andrei.broder@av.com

Abstract. A family of permutations $\mathcal{F} \subseteq S_n$ (the symmetric group) is called *min-wise independent* if for any set $X \subseteq [n]$ and any $x \in X$, when a permutation π is chosen at random in $\mathcal{F}$ we have

$$\Pr\bigl(\min\{\pi(X)\} = \pi(x)\bigr) = \frac{1}{|X|}.$$

In other words we require that all the elements of any fixed set X have an equal chance to become the minimum element of the image of X under π.

The rigorous study of such families was instigated by the fact that such a family (under some relaxations) is essential to the algorithm used by the AltaVista Web indexing software to detect and filter near-duplicate documents. The insights gained from theoretical investigations led to practical changes, which in turn inspired new mathematical inquiries and results.

This talk will review the current research in this area and will trace the interplay of theory and practice that motivated it.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, p. 808, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Testing Acyclicity of Directed Graphs in Sublinear Time

Michael A. Bender[1] and Dana Ron[2]

[1] Department of Computer Science, State University of New York at Stony Brook, Stony Brook, NY 11794-4400, USA. bender@cs.sunysb.edu.
[2] Department of Electrical Engineering – Systems, Tel Aviv University, Ramat Aviv, Israel. danar@eng.tau.ac.il.

Abstract. This paper initiates the study of testing properties of directed graphs. In particular, the paper considers the most basic property of directed graphs – acyclicity. Because the choice of representation affects the choice of algorithm, the two main representations of graphs are studied. For the adjacency matrix representation, most appropriate for dense graphs, a testing algorithm is developed that requires query and time complexity of $\tilde{O}(1/\epsilon^2)$, where ϵ is a distance parameter *independent* of the size of the graph. The algorithm, which can probe the adjacency matrix of the graph, accepts every graph that is acyclic, and rejects, with probability at least 2/3, every graph whose adjacency matrix should be modified in at least ϵ fraction of its entries so that it become acyclic. For the incidence list representation, most appropriate for sparse graphs, an $\Omega(|\mathrm{V}|^{1/3})$ lower bound is proved on the number of queries and the time required for testing, where V is the set of vertices in the graph. These results stand in contrast to what is known about testing acyclicity in *undirected* graphs.

1 Introduction

The Problem. Deciding whether a graph is *acyclic* is one of the basic algorithmic questions on directed graphs. It is well known that this problem can be solved by depth first search in time linear in the size of the graph. A natural generalization of this problem is asking how *close to acyclic* is a given graph. That is, what is the minimum number of edges (or vertices) that should be removed from the graph so that there are no remaining directed cycles. This problem is known as the *minimum feedback arc (or vertex) set* problem. Unfortunately, this problem is NP-hard [27] and even APX-hard [26]. Consequently researchers have developed approximation algorithms in various settings (including studying the complementary problem of the *maximum acyclic subgraph*) [13,18,33,29,9,23,3,15].

Testing Graph Properties. The field of *Testing Graph Properties* [20] suggests an alternate framework with which to study the above problem, and this is the approach that we take in this paper. A *property tester* determines whether a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$ has a given property or is *far from* having the property. More

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 809–820, 2000.
© Springer-Verlag Berlin Heidelberg 2000

formally, Testing Graph Properties is the study of the following family of tasks. Let P be a predetermined graph property (such as acyclicity, connectivity, or 3-colorability). A testing algorithm for property P is given *access* to a graph G so it can *query* the incidence relationships between vertices. If G has property P then the algorithms should accept with probability at least 2/3. If many edge modifications should be performed so that G has the property, then the algorithm should reject with probability at least 2/3. The success probability of the algorithm can clearly be amplified by repetitions to be arbitrarily close to 1.

We thus relax the task of deciding *exactly* whether the graph has the property, but expect of the algorithm to perform its task by observing as few vertices and edges in the graph as possible. Specifically, we are only willing to spend time that is *sub-linear in* or even *independent of* the size of the graph. Thus, in contrast to the standard notion of graph algorithms, property testing algorithms are not provided the whole graph and required to run in time polynomial in the size of the graph. Rather, they are provided access to the graph and are expected to run in sub-linear time.

More concretely, in this paper we study the question of whether a graph G is acyclic or far from acyclic. If the graph is far from acyclic (that is, many edges should be removed so that no cycle remains), then the tester should reject; if the graph actually *is* acyclic, the tester should accept; if the graph is *nearly* acyclic, then the tester may answer either way. Thus, we excuse the tester from answering the most difficult instances correctly, but we require the tester to execute much more quickly then any exact decision algorithm.

Alternate Notion of Approximation. In view of the above, property testing suggests an alternative notion of approximation that is related to the notion of *dual approximation* [24,25]. An approximation algorithm is a mechanism that trades *accuracy* for *speed*. Given an optimization problem that associates costs with solutions, the more standard notion of approximation is to find a solution that is *close* to the cost of the optimal solution. By "close," we mean that the value found is within some multiplicative factor of the optimal cost.

A property tester also trades accuracy for speed, but may use a different notion of distance. Specifically, distance is measured in terms of the number of *edge insertions* and *deletions* necessary to obtain a particular property (which, in particular, may be having a solution with a given cost).

The following example illustrates the two notions of distance. A graph G might be *nearly* 3-colorable in the sense that there is a 3-colorable graph G′ at small edit distance to G, but *far from* 3-colorable in the sense that many colors are required to color G. Alternatively, a graph G might be nearly 3-colorable in the sense that it is 4-colorable, but far from 3-colorable in the sense that no graphs having small edit distance to G are 3-colorable. Both notions are natural and the preferred choice depends on the context. In some cases the two notions coincide (e.g., Max-Cut [20]).[1]

[1] In the case of acyclicity, the notion of distance in the context of property testing and the cost approximated in the minimum feedback arc set problem are in fact the

Applications. A graph-property tester may be employed in several contexts. (1) A fast property tester can be used to speed up the slow exact decision procedure as follows. Before running the slow decision procedure, run the tester. If the fast inexact tester rejects, then we know with high confidence that the property does not hold and it is unnecessary to run the slow tester. In fact, it is often the case that when the testing algorithm rejects, it provides a *witness* that the graph does not have the property (in our case, a cycle). If the fast inexact tester accepts, then a slow exact decision procedure will determine whether the property is close to holding or actually holds. (2) There are circumstances in which knowing that a property nearly holds is good enough and consequently exact decision is unnecessary. (3) It may even be NP-hard to answer the question exactly, and so some form of approximation is inevitable.

Impact of Graph Representation. We now define precisely the notion of distance and how the tester actually probes the graph. In fact, there are two traditional representations for graphs, *adjacency matrices* and *incidence lists*. The choice of representation strongly affects these issues, as well as the applicable algorithmic techniques. We summarize the properties of each representation here.

- *Adjacency-Matrix Model.* Goldreich, Goldwasser, and Ron [20] consider the adjacency-matrix representation of graphs, where the testing algorithm is allowed to probe into the matrix. That is, the algorithm can query whether there is an edge between any two vertices of its choice. In the undirected case the matrix is symmetric, whereas in the directed case it may not be. In this representation the distance between graphs is the *fraction of entries* in the adjacency matrix on which the two graphs differ. By this definition, for a given distance parameter ϵ, the algorithm should reject every graph that requires more than $\epsilon \cdot |\mathrm{V}|^2$ edge modifications in order to acquire the tested property. This representation is most appropriate for dense graphs, and the results for testing in this model are most meaningful for such graphs.
- *(Bounded-Length) Incidence-Lists Model.* Goldreich and Ron [21] consider the incidence-lists representation of graphs. In this model, graphs are represented by lists of *length* d, where d is a bound on the degree of the graph. Here the testing algorithm can query, for every vertex v and index $i \in \{1, \ldots, d\}$, which is the i'th neighbor of v. If no such neighbor exists then the answer is '0'. In the case of directed graphs each such list corresponds to the outgoing edges from a vertex.[2] Analogously to the adjacency matrix model, the distance between graphs is defined to be the fraction of entries on which the graphs differ according to this representation. Since the total number of

same. However, the two problems do differ mainly because the former is a "promise" problem and so nothing is required of the algorithm in case the graph is close to acyclic.

[2] Actually, the lower bound we prove for this model holds also when the algorithm can query about the *incoming* edges to each vertex (where the number of incoming edges is bounded as well). We note that allowing to query about incoming edges can make testing strictly easier.

incidence-list entries is $d \cdot |\mathrm{V}|$, a graph should be rejected if the number of edges modifications required in order to obtain the property is greater than $\epsilon \cdot d|\mathrm{V}|$. [3]

Testing Directed Graphs. This paper studies property testing for *directed graphs.* Typically, a given problem on a directed graph is more difficult than the same problem on an undirected graph. In particular, testing acyclicity of undirected graphs in the adjacency-matrix representation is straightforward: Assume $\epsilon \geq \frac{2}{|\mathrm{V}|}$ (or otherwise it is possible to make an exact decision in time polynomial in $1/\epsilon$ by looking at the whole graph). The basic observation is that *any* graph having at most $\epsilon|\mathrm{V}|^2$ edges is nearly acyclic (because it is ϵ-close to the empty graph), while *only* very sparse graphs (having at most $|\mathrm{V}| - 1 < \frac{\epsilon}{2}|\mathrm{V}|^2$ edges) may be acyclic. Hence the algorithm can estimate the number of edges in the graph by sampling, and accept or reject based on this estimate. Testing acyclicity of undirected graphs in the incidence-list (bounded-degree) representation, is more interesting and is studied in [21]. However, this result does not extend to testing directed graphs.

Our Results. We first consider the problem of testing acyclicity in the adjacency matrix representation. We describe a tester whose query complexity and running time are *independent* of the size of the graph and *polynomial* in the given distance parameter ϵ. Specifically, the query complexity and running time are both $O\left(\frac{\log^2(1/\epsilon)}{\epsilon^2}\right)$. As mentioned above, the algorithm works by randomly and uniformly selecting a set of $\tilde{O}(1/\epsilon)$ vertices, and verifying whether the small subgraph induced by these vertices is acyclic. Thus, an acyclic graph is always accepted, and for all rejected graphs, the algorithms provides a "witness" that the graph is not acyclic in the form of a short cycle. A key (combinatorial) lemma used in proving the correctness of the algorithms shows that a graph that is far from acyclic contains a relatively large subset of vertices for which every vertex in the subset has many outgoing edges extending to other vertices in the subset. We then show that a sample of vertices from within this subset likely induces a subgraph that contains a cycle.

We next turn to the problem of acyclicity testing in the incidence-lists representation. We demonstrate that the problem is significantly harder in this setting. Specifically, we show an $\Omega(|\mathrm{V}|^{1/3})$ lower bound on the number of queries required for testing in this setting. To prove the bound we define two classes of directed graphs – one containing only acyclic graphs and one containing mostly graphs that are far from acyclic. We show that $\Omega(|\mathrm{V}|^{1/3})$ queries are required in order to determine from which class a randomly selected graph was chosen.

It appears that the techniques used in testing undirected graphs in the incidence-lists representation, cannot be applied directly to obtain an efficient acyclicity testing algorithm for directed graphs. Consider a graph that contains

[3] A variant of the above model allows the incidence lists to be of variant lengths [30]. In such a case, the distance is defined with respect to the total number of edges in the graph.

a relatively large subgraph that is far from acyclic, but such that many edges connect this subgraph to acyclic regions of the graph. By our lower bound, any testing algorithm should perform many queries concerning edges within this subgraph (to distinguish it from the case in which the subgraph is acyclic). However, both exhaustive local searches and random walks will "carry the algorithm away" to the acyclic regions of the graph. It would be interesting to develop an acyclicity tester that uses $O(|\mathrm{V}|^{1-\alpha})$ queries, for *any* $\alpha > 0$.

Testing Other Properties of Directed Graphs. As noted in [20, Subsection 10.1.2], some of the properties studied in that paper (in the adjacency matrix model) have analogies in directed graphs. Furthermore, the algorithms for testing these properties can be extended to directed graphs. In particular, these properties are defined by partitions of the vertices in the graph with certain constraints on the sizes of the sets in the partition as well as on the density of edges between these sets. The techniques of [1] (for testing properties of undirected graphs in the adjacency-matrix representation), can also be extended to testing properties of directed graphs (Private communications with Noga Alon).

Another basic property of directed graphs, is (*strong*) *connectivity*. Namely, a directed graph is strongly connected if there is a directed path in the graph from any vertex to any other vertex. Testing this property is most meaningful in the incidence-lists model, as every graph can be made strongly connected by adding at most $2N$ directed edges. The undirected version of this problem is studied in [21], where an algorithm having query and times complexities $\tilde{O}(1/\epsilon)$ is presented and analyzed. As we show in the long version of this paper [8], this algorithm can be extended to the directed case *if the algorithm can also perform queries about the incoming edges to each vertex*. Otherwise, (the algorithm can only perform queries about outgoing edges), a lower bound of $\sqrt{N}$ on the number of queries can be obtained.

Related Work. Property testing of functions was first explicitly defined in [32] and extended in [20]. Testing algebraic properties (e.g., linearity or being a polynomial of low-degree) plays an important role in the settings of Program Testing (e.g., [10,32,31]) and Probabilistically-Checkable Proof systems (e.g., [7,6,14,5,4]). As mentioned previously, the study of testing graph properties was initiated in [20], where, in particular, the adjacency-matrix model was considered. Some of the properties studied in that work are bipartitness, k-colorability, having a clique of a certain size and more. In [21], testing properties of graphs represented by their incidence lists was considered. Some of the the properties studied in that work are k-connectivity and acyclicity.

Ergun *et. al.* [12] give a poly$(1/\epsilon)$-time algorithm for testing whether a relation is a total order. This can viewed as a special case of testing acyclicity in the adjacency-matrix model, where it is assumed that a directed edge exists between *every* two vertices.

Other papers concerning testing of graph properties and other combinatorial properties include [22,12,28,19,11]. Recently, Alon *et. al.* [1] presented a general family of graph properties that can be tested using a sample that is independent

of the size of the graph (though the dependence on the distance parameter ϵ is somewhat high). In [2] it is shown that all properties defined by regular languages can be tested using a sample of size almost linear in the distance parameter.

As mentioned previously, the related minimum feedback set problem is APX-Hard [26], and its complementary, maximum acyclic subgraph is APX-Complete [29]. The former can be approximated to within a factor of $O(\log|\mathrm{V}|\log\log|\mathrm{V}|)$ [13], and the latter to within $2/(1+\Omega(1/\sqrt{\Delta}))$ where Δ is the maximum degree [9,23]. Variants of these problems are studied in the following papers [33,18,3]. Perhaps the result most closely related to our work is that of Frieze and Kannan [15]. They show how to approximate the size of the maximum acyclic subgraph to within an additive factor of $\epsilon|\mathrm{V}|^2$, in time exponential in $1/\epsilon$. In comparison to their result, we solve a more restricted problem in time polynomial in $1/\epsilon$ as opposed to exponential. In addition, since our analysis is tailored to the particular problem (as opposed to theirs which follows from a general paradigm that gives rise a family of approximation algorithms), it may give more insight into the problem in question.

With the current trend of increasing memory and storage sizes, the problem of examining large structures in sublinear time has been studied in other contexts. For example, Gibbons and Matias [16,17] develop a variety of data structures that glean information from large databases so they can be examined in sublinear time.

2 Definitions

Let $\mathrm{G}=(\mathrm{V},\mathrm{E})$ be a directed graph, where $|\mathrm{V}|=N$, and $\mathrm{E}\subseteq\mathrm{V}\times\mathrm{V}$ consists of *ordered* pairs of vertices. For a given set of vertices $\mathrm{U}\subseteq\mathrm{V}$, let $\mathrm{G}(\mathrm{U})$ denote the subgraph of G induced by U, and for any two sets of vertices U_1 and U_2, let $\mathrm{E}(\mathrm{U}_1,\mathrm{U}_2)$ denote the set of edges going from vertices in U_1 to vertices in U_2. That is, $\mathrm{E}(\mathrm{U}_1,\mathrm{U}_2)\stackrel{\text{def}}{=}\{(u_1,u_2)\in\mathrm{E}:\ u_1\in\mathrm{U}_1,\ u_2\in\mathrm{U}_2\}$.

We say that a graph G is *acyclic* if it contains no directed cycles. In other words, G is acyclic if and only if there exists a (one-to-one) ordering function $\phi:\mathrm{V}\mapsto\{1,\ldots,N\}$, such that for every $(v,u)\in\mathrm{E}$, $\phi(v)<\phi(u)$. We say that an edge $(v,u)\in\mathrm{E}$ is a *violating edge* with respect to an ordering $\phi(\cdot)$, if $\phi(v)>\phi(u)$.

We consider two representations of (directed) graphs. In the *adjacency-matrix* representation, a graph G is represented by a 0/1 valued $N\times N$ matrix M_G, where for every pair of vertices $u,v\in\mathrm{V}$, $\mathrm{M}_\mathrm{G}[u,v]=1$ if and only if $(u,v)\in\mathrm{E}$. This representation is more appropriate for dense graphs than sparse graphs, because with sparse graphs the representation entails a large space wastage. In the *incidence-lists* representation, a graph G is represented by an $N\times d$ matrix L_G, (which can be viewed as N *lists*), where d is a bound on the outdegree of each vertex in G. For $v\in\mathrm{V}$ and $i\in[d]$, $\mathrm{L}_\mathrm{G}[v,i]=u$, if an only if the i'th edge going out of v is directed to u. If such an edge does not exist then the value of the entry is '0'.

For any $0\le\epsilon\le1$, a graph G in either of the two representations, is said to be ϵ-*close to acyclic*, if at most an ϵ-fraction of entries in G's representation

need to be modified to make G acyclic. If *more* than an ϵ fraction of entries must be modified, than it is *ϵ-far from acyclic*. Because the adjacency-matrix representation has size N^2, this means that a graph G in the adjacency-matrix representation is ϵ-close to being acyclic if at most $\epsilon \cdot N^2$ edges can be removed to make G acyclic. Because the incidence-lists representation has size $d \cdot N$, the number of edges that should be removed in this representation is at most $\epsilon \cdot dN$. Note that a graph is ϵ-close to acyclic if and only if there exists an order function $\phi(\cdot)$, with respect to which there are at most ϵN^2 (similarly, $\epsilon \cdot dN$), violating edges. We say in this case that $\phi(\cdot)$ is *ϵ-good*.

A testing algorithm for acyclicity is given a distance parameter ϵ, and oracle access to an unknown graph G. In the adjacency-matrix representation this means that the algorithm can query for any two vertices u and v whether $(u, v) \in$ E. In the incidence-lists representation this means that the algorithm can query, for any vertex v and index $i \in [d]$, what vertex does the i'th edge going out of v point to. If the graph G is acyclic then the algorithm should accept with probability at least 2/3, and if it is ϵ-far from acyclic then the algorithm should reject with probability at least 2/3.

3 Testing Acyclicity in the Adjacency-Matrix Representation

We next give our algorithm for testing acyclicity when the graph is represented by its adjacency matrix. Similarly to several previous testing algorithms in the (undirected) adjacency-matrix model, the algorithm is the "natural" one. Namely, it selects a random subgraph of G (having only $\tilde{O}(1/\epsilon)$ vertices), and checks whether this subgraph is acyclic (in which case it accepts) or not (in which case it rejects). Observe that the sample size is independent of the size of G.

Acyclicity Testing Algorithm

1. Uniformly and independently select a set of $\Theta(\log(1/\epsilon)/\epsilon)$ vertices and denote the set by U.
2. For every pair of vertices $v_1, v_2 \in$ U, query whether either $(v_1, v_2) \in$ E or $(v_2, v_1) \in$ E, thus obtaining the subgraph G(U) induced by U.
3. If G(U) contains a cycle, then *reject*, otherwise *accept*.

Theorem 1 *The algorithm described above is a testing algorithm for acyclicity having query and time complexity $\tilde{O}(1/\epsilon^2)$. Furthermore, if the graph* G *is acyclic it is always accepted, and whenever the algorithm rejects a graph it provides a certificate of the graph's cyclicity (in form of a short cycle).*

The bound on the query and time complexity of the algorithm follows directly from the description of the algorithm. In particular, there are $O(\log^2(1/\epsilon)/\epsilon^2)$ pairs of vertices in U, which limits the number of queries made as well as the number of edges in G(U). To verify whether G(U) is acyclic or not, a Breadth-First-Search (BFS) can be performed starting from any vertex in G(U). The

time complexity of this search is bounded by the number of edges in G(U), as desired. The second statement in the theorem is immediate as well. It remains to be shown that every graph that is ϵ-far from being acyclic is rejected with probability at least 2/3.

Proof Idea. We prove Theorem 1 using two lemmas. Lemma 2 shows that if a graph G is far from acyclic, then G contains a relatively large set W such that each vertex in W has many outgoing edges to other vertices in W.[4] Lemma 3 shows that if we uniformly select a sufficient number of vertices from W, then with probability at least 9/10 the underlying graph induced by these vertices contains a cycle. To prove Theorem 1, we show that with sufficiently high probability, a large enough sample of vertices in G contains enough vertices in W to find a cycle with the desired probability.

Definitions. To formalize the above ideas, we use the following definitions. For any vertex $v \in \mathrm{V}$, let $\mathrm{O}(v) \stackrel{\text{def}}{=} \{u : (v,u) \in \mathrm{E}\}$ be the set of v's outgoing edges. Given a set $\mathrm{W} \subseteq \mathrm{V}$, we say that v has *low outdegree with respect to* W, if $|\mathrm{O}(v) \cap \mathrm{W}| \leq \frac{\epsilon}{2}N$; otherwise it has *high outdegree with respect to* W.

Lemma 2 *If* G *is ϵ-far from acyclic, then there exists a set* $\mathrm{W} \subseteq \mathrm{V}$*, such that* $|\mathrm{W}| \geq \sqrt{\frac{\epsilon}{2}}N$*, and every* $v \in \mathrm{W}$ *has high outdegree with respect to* W.

Lemma 3 *Let* $\mathrm{W} \subseteq \mathrm{V}$ *be a set of vertices such that for every* $v \in \mathrm{W}$*,* $|\mathrm{O}(v) \cap \mathrm{W}| \geq \epsilon'|\mathrm{W}|$ *for some* $\epsilon' > 0$*. Suppose we uniformly and independently select* $\Omega(\log(1/\epsilon')/\epsilon')$ *vertices in* W*. Then with probability at least* 9/10 *(over this selection) the subgraph induced by these vertices contains a cycle.*

We prove the two lemmas momentarily, but first we show how Theorem 1 follows from the two lemmas.

Proof of Theorem 1: If G is acyclic, then clearly it always passes the test. Thus, consider the case in which G is ϵ-far from acyclic. By Lemma 2, there exists a set $\mathrm{W} \subseteq \mathrm{V}$, such that $|\mathrm{W}| \geq \sqrt{\frac{\epsilon}{2}}N$, and every $v \in \mathrm{W}$ has high outdegree with respect to W. Let $\alpha \stackrel{\text{def}}{=} |\mathrm{W}|/N$ be the *fraction of graph vertices that belong to* W, so that $\alpha \geq \sqrt{\frac{\epsilon}{2}}$. By applying a (multiplicative) Chernoff bound we have that for every integer $m > 12$, with probability at least 9/10, a uniformly and independently selected sample of $2m/\alpha$ vertices contains at least m (not necessarily distinct) vertices in W (where these vertices are uniformly distributed in W). Assume this is in fact the case (where we account for the probability of error and set m below).

Let $\epsilon' \stackrel{\text{def}}{=} \frac{\epsilon}{2\alpha}$ so that by definition of α, for every $v \in \mathrm{W}$, $|\mathrm{O}(v) \cap \mathrm{W}| \geq \epsilon'|\mathrm{W}|$. By setting the quantity m to be $\Theta(\log(1/\epsilon')/\epsilon')$, and applying Lemma 3, we obtain that conditioned on there being m elements in the sample that belong to W,

[4] In fact, as we showed in an earlier version of this paper, there exists a relatively large set W such that every vertex in W also has many incoming edges from other vertices in W. However, we no longer need this stronger property.

a cycle is observed with probability at least 9/10. Adding the two error probabilities, and noting that the total size of the sample is $O(m/\alpha) = O(\log(1/\epsilon)/\epsilon)$, the theorem follows. □

Now that we have demonstrated how Theorem 1 follows from Lemmas 2 and 3, we prove the lemmas themselves.

Proof of Lemma 2: We prove the *contrapositive* of the statement in the lemma: If such a set W does not exist, then the graph is ϵ-close to being acyclic. In other words, we show that if for every subset $\mathrm{Z} \subseteq \mathrm{V}$ having size at least $\sqrt{\frac{\epsilon}{2}}N$, there exists at least one vertex in Z having small outdegree with respect to Z, then the following holds. There exists an order $\phi : \mathrm{V} \mapsto [N]$, and a set of edges T of size at most ϵN^2, such that the edges of T are the only violating edges with respect to ϕ.

We define ϕ and construct T in N steps. At each step we select a vertex v for which ϕ is not yet determined, and set the value of $\phi(v)$. We maintain an index ℓ (*last*), where initially $\ell = N$. At the start of a given step, let $\mathrm{Z} \subseteq \mathrm{V}$ denote the set of vertices for which ϕ is yet undefined (where initially, $\mathrm{Z} = \mathrm{V}$). As long as $|\mathrm{Z}| \geq \sqrt{\frac{\epsilon}{2}}N$, we do the following. Consider any vertex v that has low outdegree with respect to Z (where the existence of such a vertex is ensured by our (counter) assumption). Then we set $\phi(v) = \ell$, decrease ℓ by 1, and let $\mathrm{T} \leftarrow \mathrm{T} \cup \{(v, u) \in \mathrm{E} : \ u \in \mathrm{Z}\}$. Hence, at each step, the size of T increases by at most $\frac{\epsilon}{2}N$.

Finally, when $|\mathrm{Z}| < \sqrt{\frac{\epsilon}{2}}N$, so that the vertices in Z may all have high outdegree with respect to Z, we order the vertices in Z arbitrarily between 1 and ℓ, and add to T all (at most $|\mathrm{Z}|^2 < \frac{\epsilon}{2}N^2$) edges between vertices in Z. Thus, the total number of edges in T is bounded by ϵN^2, as desired.

It remains to show that there are no other violating edges with respect to ϕ. Namely, for every $(v, u) \in \mathrm{E} \setminus \mathrm{T}$, it holds that $\phi(v) < \phi(u)$. Consider any such edge $(v, u) \in \mathrm{E} \setminus \mathrm{T}$. We claim that necessarily the value of ϕ was first defined for u, implying that in fact $\phi(v) < \phi(u)$ (since the value ℓ given by ϕ decreases as the above process progresses). This must be the case since otherwise, if the value of ϕ was first defined for v then the edge (v, u) would have been added to T, contradicting our assumption that $(v, u) \in \mathrm{E} \setminus \mathrm{T}$. □

Proof of Lemma 3: Let $m = \frac{c \cdot \ln(1/\epsilon')}{\epsilon'} + 1$ be the number of vertices selected (uniformly and independently) from W, where c is a constant that is set below. We shall show that with probability at least 9/10 over the choice of such a sample U, for every vertex $v \in \mathrm{U}$, there is another vertex $u \in \mathrm{U}$ such that $(v, u) \in \mathrm{E}$. This implies that the subgraph induced by U has no sink vertex, and hence contains a cycle.

Let the m vertices selected in the sample U be denoted $v_1, \ldots, v_m$. For each index $1 \leq i \leq m$, let $\mathcal{E}_i$ be the event that there exists a vertex v_j such that $(v_i, v_j) \in \mathrm{E}$. In other words, $\mathcal{E}_i$ is the (desirable) event that v_i has non-zero outdegree in the subgraph induced by U. We are interested in upper bounding the probability that for some i the event $\mathcal{E}_i$ does not hold. That is, $\Pr\left[\bigcup_{i=1}^{m} \neg\mathcal{E}_i\right]$.

Since the vertices in the sample are chosen uniformly and independently, and every vertex in W has outdegree at least $\epsilon' \cdot |\mathrm{W}|$, for each fixed i,

$$\Pr[\neg\mathcal{E}_i] \;\leq\; (1-\epsilon')^{m-1} \;<\; \exp(-(m-1)\epsilon') \;=\; \exp(-c\ln(1/\epsilon')) \;=\; (\epsilon')^{-c} \quad (1)$$

By applying a probability union bound,

$$\Pr\left[\bigcup_{i=1}^{m}\neg\mathcal{E}_i\right] \;\leq\; \sum_{i=1}^{m}\Pr[\neg\mathcal{E}_i] \;<\; \left(\frac{c\cdot\ln(1/\epsilon')}{\epsilon'}+1\right)\cdot(\epsilon')^{-c} \quad (2)$$

Setting c to be a sufficiently large constant (say, $c \geq 10$), for any $\epsilon' \leq 1/2$ the above probability is at most $1/10$ as required. □

4 Testing Acyclicity in the Incidence-Lists Representation

In this section we give a lower bound of $\Omega(N^{1/3})$ for testing acyclicity in the incidence-lists representation, when d and ϵ are constant. This lower bounds holds even when the algorithm may query about the incoming edges to each vertex (where the indegree of each vertex is also at most d).

Theorem 4 *Testing Acyclicity in the incidence-lists representation with distance parameter $\epsilon \leq \frac{1}{16}$, requires more than $\frac{1}{4}\cdot N^{1/3}$ queries.*

Due to space limitations, the full proof appears in the long version of this paper [8], and here we just sketch the idea. To prove the bound we define two classes of (directed) graphs, $\mathcal{G}_1$ and $\mathcal{G}_2$, each over N vertices, with degree bound d. All graphs in $\mathcal{G}_1$ are acyclic, and we show that almost all graphs in $\mathcal{G}_2$ are ϵ-far from acyclic (for $\epsilon = \frac{1}{16}$). We then prove that no algorithm can distinguish between a graph chosen randomly in $\mathcal{G}_1$ and a graph chosen randomly in $\mathcal{G}_2$ in less than $\alpha \cdot N^{1/3}$ queries, for $0 < \alpha \leq \frac{1}{4}$. The classes are defined as follows.

- Each graph in $\mathcal{G}_1$ consists of $K = N^{1/3}$ layers, $\mathrm{L}_1, \ldots, \mathrm{L}_K$, each having $M = N^{2/3}$ vertices. From each layer L_i there are $d \cdot |\mathrm{L}_i| = d \cdot M$ edges going to layer L_{i+1}, where these edges are determined by d matchings between the the vertices in the two layers.
- Each graph in $\mathcal{G}_2$ consists of two equal-size subsets of vertices, S_1 and S_2. There are $d \cdot \frac{N}{2}$ edges going from S_2 to S_1, and $d \cdot \frac{N}{2}$ edges going from S_1 to S_2. The two sets of edges are each defined by d matching between S_1 and S_2.

In both cases, every edge has the same label at both its ends (determined by the matching).

Acknowledgements

We would like to thank an anonymous program committee member for helping us simplify the proof of Theorem 1.

References

1. N. Alon, E. Fischer, M. Krivelevich, and M Szegedy. Efficient testing of large graphs. In *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science*, pages 656–666, 1999.
2. N. Alon, M. Krivelevich, I. Newman, and M Szegedy. Regular languages are testable with a constant number of queries. In *Proceedings of the Fortieth Annual Symposium on Foundations of Computer Science*, pages 645–655, 1999.
3. S. Arora, A. Frieze, and H. Kaplan. A new rounding procedure for the assignment problem with applications to dense graph arrangement problems. In *37th Annual Symposium on Foundations of Computer Science*, pages 21–30. IEEE, 14–16 October 1996.
4. S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proceedings of the Thirty-Third Annual Symposium on Foundations of Computer Science*, pages 14–23, 1992.
5. S. Arora and S. Safra. Probabilistic checkable proofs: A new characterization of NP. In *Proceedings of the Thirty-Third Annual Symposium on Foundations of Computer Science*, pages 1–13, 1992.
6. L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991.
7. L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.
8. M. Bender and D. Ron. Testing acyclicity of directed graphs in sublinear time. Full version of this paper. Available from `http://www.eng.tau.ac.il/~danar`, 2000.
9. B. Berger and P. W. Shor. Tight bounds for the maximum acyclic subgraph problem. *Journal of Algorithms*, 25(1):1–18, October 1997.
10. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of the Association for Computing Machinery*, 47:549–595, 1993.
11. Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonocity. In *Proceedings of RANDOM99*, 1999.
12. F. Ergun, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, pages 259–268, 1998.
13. G. Even, J. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20, 1998.
14. U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. *Journal of the Association for Computing Machinery*, 43(2):268–292, 1996.
15. A. Frieze and R. Kannan. Quick approximations of matrices. An extended abstract of some of this work appeared in FOCS96 under the title: The Regularity Lemma and approximation schemes for dense problems, 1997.
16. P. B. Gibbons and Y. Matias. New sampling-based summary statistics for improving approximate query answers. *SIGMOD Record: Proc. ACM SIGMOD Int. Conf. Management of Data*, 27(2):331–342, 2–4 June 1998.
17. P. B. Gibbons and Y. Matias. Synopsis data structures for massive data sets. *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science: Special Issue on External Memory Algorithms and Visualization*, A, 1999. to appear.

18. M. Goemans and D. Williamson. Primal-dual approximation algorithms for feedback problems in planar graphs. *Combinatorica*, 18, 1998.
19. O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. To appear in *Combinatorica*. A preliminary (and weaker) version of this work appeared in FOCS98, 1999.
20. O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the Association for Computing Machinery*, 45(4):653–750, 1998. An extended abstract appeared in FOCS96.
21. O. Goldreich and D. Ron. Property testing in bounded degree graphs. In *Proceedings of the Thirty-First Annual ACM Symposium on the Theory of Computing*, pages 406–415, 1997.
22. O. Goldreich and D. Ron. A sublinear bipartite tester for bounded degree graphs. In *Proceedings of the Thirty-Second Annual ACM Symposium on the Theory of Computing*, 1998. To appear in *Combinatorica*.
23. R. Hassin and S. Rubinstein. Approximations for the maximum acyclic subgraph problem. *Information Processing Letters*, 51(3):133–140, August 1994.
24. D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the Association for Computing Machinery*, 34(1):144–162, January 1987.
25. D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.
26. V. Kann. *On the Approximability of NP-Complete Optimization Problems*. PhD thesis, Department of Numberical Analysis and Computer Science, Royal Institute of Technology, Stockholm, 1992.
27. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.
28. M. Kearns and D. Ron. Testing problems with sub-learning sample complexity. In *Proceedings of the Eleventh Annual ACM Conference on Computational Learning Theory*, pages 268–277, 1998.
29. C. Papadimitriou and M. Yannakakis. Optimization, approximization and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.
30. M. Parnas and D. Ron. Testing the diameter of graphs. In *Proceedings of Random99*, pages 85–96, 1999.
31. R. Rubinfeld. Robust functional equations and their applications to program testing. In *Proceedings of the Thirty-Fifth Annual Symposium on Foundations of Computer Science*, 1994.
32. R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.
33. P. D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15, 1995.

Computing the Girth of a Planar Graph *

Hristo N. Djidjev[1]

Department of Computer Science
University of Warwick
Coventry CV4 7AL, UK

Abstract. The girth of a graph G has been defined as the length of a shortest cycle of G. We design an $O(n^{5/4} \log n)$ algorithm for finding the girth of an undirected n-vertex planar graph, giving the first $o(n^2)$ algorithm for this problem. Our approach combines several techniques such as graph separation, hammock decomposition, covering of a planar graph with graphs of small tree-width, and dynamic shortest path computation. We discuss extensions and generalizations of our result.

1 Introduction

Given an (unweighted) graph G, the *length* of a path p in G is the number of the edges in p. The *girth* of G (denoted by $girth(G)$) was defined by Harary [15] as the length of a shortest cycle in G, or infinity if G has no cycle. The girth is a basic combinatorial characteristic of graphs and its relations to other graph properties have been extensively studied. In particular, Erdös [12], Lovasz [19], Bollobás [4], Cook [5], and others studied the relationship between the girth and the chromatic number of a graph. Thomassen [22] and Mader [20] studied the relationship between the girth and the existence of certain type of minors of the graph. Other results relate the girth of the graph to the minimum or the average degrees of its vertices, its diameter, its maximum genus, and its connectivity (see [6]).

The first efficient algorithm for computing the girth of graph was given by Itai and Rodeh [16], who describe an $O(nm)$ algorithm for computing the girth of a general n-vertex m-edge graph G. They also design an $O(n^2)$ algorithm for computing the girth within an additive error of one. Finding shortest cycles of even and odd lengths have been studied by Monien [21] and Vazirani and Yannakakis [23]. There are numerous results on finding a cycle of a given length in general or special graphs; see e.g. [1] and [2] for recent results and references.

In the case of planar graphs Itai and Rodeh [16] give an $O(n)$ algorithm for finding a triangle in the graph, if one exists (and thus solves the girth problem for planar graphs in case $girth(G) \leq 3$ in $O(n)$ time). Their results were generalized by Eppstein [11], who developed an $O(n)$ algorithm for finding the girth of a planar graph G provided $girth(G) = O(1)$. (His algorithm, however, is superexponential with respect to $girth(G)$.)

* This work was partially supported by the EPA grant R82-5207-01-0, EPSRC grant GR/M60750, and RTDF grant 98/99-0140.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 821-831, 2000
© Springer-Verlag Berlin Heidelberg 2000

Note that, for an embedded planar graph G, the size of each face of G is an upper bound on the girth of G. However, the shortest cycle of G does not necessarily needs to be a face of G.

In this paper we present an algorithm that finds the girth of an arbitrary undirected n-vertex planar graph in $O(n^{5/4} \log n)$ time.

Our approach makes use of recently developed fast dynamic algorithms for computing shortest paths in planar graphs with small face-on-vertex cover [7]. If G has an appropriately large girth, then we show that the shortest cycle in G can be computed by combining separator based divide-and-conquer with the dynamic shortest path algorithm from [7]. If, on the other hand, the girth of G is very small, then we can decompose G into subgraphs of small diameter and such that any shortest cycle of G is contained also in one of the subgraphs of the decomposition. Therefore, we can search for a shortest cycle in each subgraph independently, which will be less expensive in terms of total computation time.

The rest of the paper is organized as follows. In Section 2, we give some definitions and review basic facts related to graph separators. In Section 3 and Section 4, we describe algorithms efficient for graphs of large or small girth, respectively. In Section 5, we describe the algorithm for the case of general graphs. In the last section we discuss some related results and open problems.

2 Preliminaries

By $G = (V, E)$ we denote in the rest of this paper an undirected connected graph with n vertices. Given an edge $e \in E$ and a set $S \subset V$, by $G - e$ we denote the graph $(V, E \setminus \{e\})$ and by $G - S$ we denote the graph $(V \setminus S, E \setminus (S \times S))$. By $deg(G)$ we denote the maximum degree of a vertex of G.

If any edge e of G has a non-negative cost $cost(e)$ associated with it, then the *length* of a path p of G is the sum of the costs of all edges of p. The *distance* between two vertices x and y of G is the minimum length of a path joining x and y. The *single-source shortest path problem* asks, given a source vertex s of G, to compute the distances between s and all other vertices of G. If G is planar, then the single-source shortest path problem for G can be solved in $O(n)$ time [17].

The graph G is *planar* if G can be embedded in the plane so that no two edges intersect except at a common endpoint. A planar graph of n vertices has at most $3n - 3 = O(n)$ edges. A graph already embedded in the plane is a *plane graph*.

A *separator* of G is a set of vertices whose removal leaves no connected component of more than $n/2$ vertices. If G is planar, then G has a separator of size $O(\sqrt{n})$ [18, 9] and if G has genus $g > 0$, then G has a separator of $O(\sqrt{gn})$ vertices [8, 14]. In both cases, the corresponding separators can be found in $O(n)$ time.

If a graph G has non-negative weights associated with its vertices, then a *weighted separator* of G is a set of vertices whose removal leaves no connected component of weight greater than half the weight of G.

For other standard definitions from graph theory see [6] or any other textbook on graphs.

3 Finding shortest cycles in graphs of large girth

In this section we assume that the girth of G is bounded from bellow by some constant $\gamma > 0$. We will use the dynamic shortest path algorithm from [7] that runs faster for graphs of small face-on-vertex cover.

A *hammock decomposition* of an n-vertex graph G was defined by Frederickson in [13] as a decomposition of G into certain outerplanar digraphs called *hammocks*. Hammocks satisfy the following properties:

(i) each hammock has at most four vertices, called *attachment vertices*, shared with the rest of the graph;

(ii) the hammock decomposition spans all the edges of G, i.e., each edge belongs to exactly one hammock; and

(iii) the number of hammocks produced is the minimum possible (within a constant factor) among all possible decompositions.

Frederickson showed in [13] that hammock decompositions of a small cardinality can be computed fast for planar graphs if a good approximation of the minimum cardinality of a face-on-vertex cover is known.

Theorem 1. *[13] Let G be an n-vertex planar graph and let $f(G)$ be the minimum number of faces (over all embeddings of G in the plane) that cover all vertices of G. Then G can be decomposed into $O(f(G))$ hammocks in $O(n)$ time.*

The results of Frederickson [13], Djidjev et al. [7], and others have shown that several shortest paths problems can be solved more efficiently for certain classes of graphs with hammock decompositions of a small cardinality.

Our goal is to make use of the following shortest path algorithm based on the idea of a hammock decomposition.

Theorem 2. *[7] Let G be an n-vertex planar digraph with nonnegative edge costs and assume that G has a hammock decomposition of cardinality q. There exists an algorithm for the dynamic shortest path problem on G with the following performance characteristics:*

(i) preprocessing time and space $O(n)$;

(ii) query time for computing the distance between any two vertices $O(\log n + q)$;

(iii) time for updating the data structure after any edge cost modification or edge deletion $O(\log n)$.

The following algorithm for finding the girth is based on Theorem 1, Theorem 2, and the observation that the size of each face of a plane graph is not less than the girth of that graph.

First we transform G into a planar graph G' of degree at most 3 and $O(m)$ vertices, where $m = O(n)$ is the number of edges of G, by replacing each vertex of G of degree $k > 3$ by a balanced binary tree with k leaves as illustrated on Figure 1. Assign a cost $cost(e)$ to any edge e of the tree equal either to one, if e is incident to a leaf of the tree, or zero, otherwise. This transformation has the following properties:

(i) The edges of G correspond to those edges of G' that have weight one;

(ii) G and G' have the same number of faces;

(iii) Any path of G of length (i.e. number of edges since G is unweighted) l is transformed into a path of weighted length l and at most $l \log n$ edges.

Next we apply to G' the following algorithm.

Algorithm LARGE_GIRTH

{Finds the girth of a biconnected plane graph G with n vertices, f faces, maximum vertex degree 3, and nonnegative costs $cost(\cdot)$ on its edges}

1. Find a hammock decomposition of cardinality $O(f)$ of G using the algorithm from Theorem 1.
2. Preprocess G for shortest path queries using the preprocessing algorithm from Theorem 2.
3. Construct a separator S of G of size $O(\sqrt{n})$ that divides G into components of at most $n/2$ vertices each.
4. For each resulting component K, find the length of a shortest cycle in K by applying this algorithm recursively.
5. For each edge e incident to a vertex of S compute the length $c(e)$ of the shortest cycle in G containing e as follows.
   (a) Change the cost of e to $+\infty$ (which has the effect of deleting e from G) by using the update algorithm from Theorem 2.
   (b) Compute the distance $c'(e)$ between the endpoints of e in the modified graph using the query algorithm from Theorem 2.
   (c) Assign $c(e) := c'(e) + cost(e)$, where $cost(e)$ denotes the original cost of e.
   (d) Assign to e its original cost by using the update algorithm from Theorem 2.
6. Return the minimum length of a cycle in G by combining the results from Steps 4 and 5.

Now we analyze the correctness of the algorithm and its running time for $\gamma = O(n^{1/2-\varepsilon})$ for some $\varepsilon > 0$, since the value of the parameter γ that will be chosen in Section 5 will satisfy that condition.

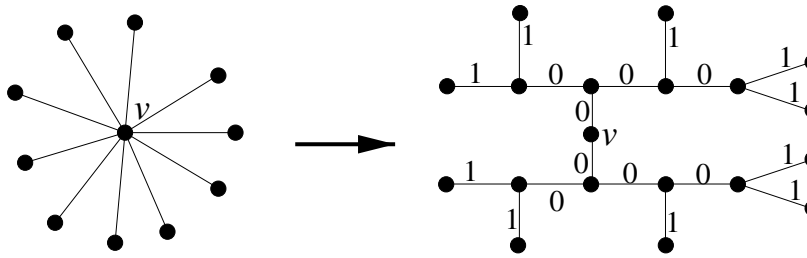


Fig. 1. Reducing the maximum vertex degree.

Lemma 1. *If G is a biconnected planar graph G of degree three and girth$(G) \geq \gamma$, then Algorithm* Large_Girth *computes the girth of G in $O(n^{3/2}/\gamma)$ time, assuming $\gamma = O(n^{1/2-\varepsilon})$ for some $\varepsilon > 0$.*

Proof: *Correctness:* Assume that c is a shortest cycle of G. If c does not contain a vertex of S, then c will belong to some component K considered in Step 4. K can not contain a cycle shorter than c since otherwise c will not be a shortest cycle of G. Hence the length of c will be correctly computed in Step 4.

If c contains a vertex from S, then c will contain an edge e incident to a vertex from S that is considered in Step 5. Since c is a shortest cycle of G, then $c - e$ will be a shortest path between the endpoints of e in $G - e$ (Figure 2). Hence in that case the length of c will be correctly computed in Step 5.

Time complexity: The time for Steps 1, 2, 3, and 6 is $O(n)$. The number of iterations of Step 5 is $O(\sqrt{n})$, since each vertex of G (and thus each vertex of S) has degree $O(1)$. The time for each iteration of Step 5 is dominated by the time of Step 5 (b), which time, according to Theorem 2 and Theorem 1, is $O(\log n + f)$. Hence the time $T(n, f)$ for Algorithm Large_Girth satisfies the recurrence

$$\begin{aligned} T(1, f) &= O(1), \\ T(n, f) &\leq \max\{T(n/2, f_1) + T(n/2, f_2) \mid f_1 + f_2 \leq f\} \\ &\quad + O(n) + O(\sqrt{n}\,(f + \log n)), \quad \text{for } n > 1. \end{aligned}$$

For $n > 1$, one can represent $T(n, f)$ as

$$T(n, f) = T_1(n) + T_2(n, f),$$

where

$$\begin{aligned} T_1(n) &\leq 2T_1(n/2) + O(n) \quad \text{and} \\ T_2(n, f) &\leq \max\{T_2(n/2, f_1) + T_2(n/2, f_2) \mid f_1 + f_2 \leq f\} \\ &\quad + O(\sqrt{n} \cdot f). \end{aligned}$$

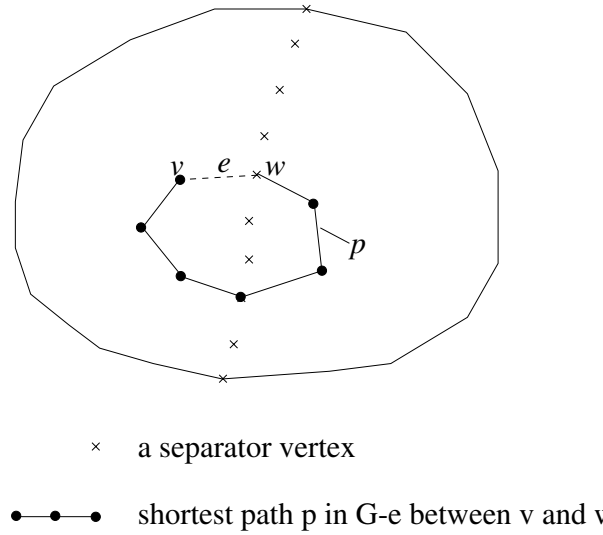


Fig. 2. An Illustration to the proof of Lemma 1. The cycle $p + e$ is a shortest cycle containing e.

By unfolding, we have for the case $n > 1$

$$\begin{aligned} T_2(n,f) &\le c\,(\sqrt{n}\cdot f + (\sqrt{n/2}\cdot f_1 + \sqrt{n/2}\cdot f_2) + \cdots) \\ &= c\,f\,(\sqrt{n} + \sqrt{n/2} + \cdots) = cfO(\sqrt{n}) \;=\; O(f\sqrt{n}). \end{aligned}$$

Clearly $T_1(n) = O(n \log n)$ and hence $T(n,f) = O(f\sqrt{n} + n\log n)$.

Finally, since G is biconnected each face of G is a cycle, which implies that the size of each face is not less than γ. Moreover, each edge of G belongs to exactly two faces of the embedding and hence $f \le 2 \cdot |E(G)| \,/\, \gamma = O(n/\gamma)$.

Thus $T(n,f) = O(f\sqrt{n} + n\log n) = O(n^{3/2}/\gamma)$ for $\gamma = O(n^{1/2-\varepsilon})$. □

4 Finding shortest cycles in graphs of small girth

In this section we assume that the girth of the input graph is smaller than certain parameter γ whose value will be determined in Section 5. For the proof of the next lemma we use a technique developed by Baker [3] and Eppstein [11].

Lemma 2. *Let G be an n-vertex planar graph and let d be any integer. Then we can find in $O(n)$ time a set of subgraphs G_i of G with the following properties:*

(i) The sum of the sizes of all subgraphs G_i is $O(n)$;

(ii) Every subgraph of G_i has a separator of size $O(d)$;

(iii) Any cycle of length at most $4d$ is contained in some of the subgraphs G_i.

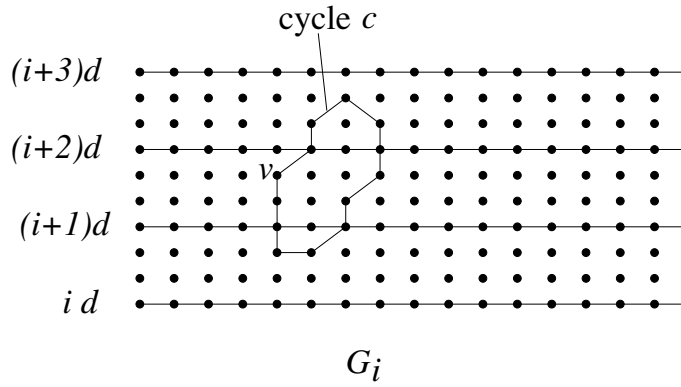


Fig. 3. An Illustration to the proof of Lemma 2.

Proof: Construct a breadth-first spanning tree T of G and divide the vertices of G into levels according to their distance to the root of T. Define the graph G_i to be the graph induced by the vertices on levels from id to $(i+3)d-1$.

Any vertex on a level between id and $(i+1)d-1$ will belong only to the subgraphs G_{i-2}, G_{i-1} and G_i (if they exist). Thus any vertex of G will belong to at most 3 of the subgraphs G_i. Property (i) follows.

In order to prove (ii), for each graph G_i define a new vertex v_i and connect it to all vertices on level id. The resulting graph G'_i has a breadth first spanning tree of radius $r = 3d$. According to a lemma from [9], each weighted planar graph with a breadth-first spanning tree of radius r has a weighted separator of no more than $3r+1$ vertices one of which is the root of the tree. Let H be any subgraph of G_i. Assign a weight 1 to each vertex of H and a weight 0 to each vertex of G'_i that is not in H. By applying the lemma from [9] on the weighted version of G'_i, we find a separator of H of size at most $9d = O(d)$.

Finally, for proving that condition (iii) holds, we note that the endpoints of any edge of G belong either to the same level, or to two consecutive levels of G. Let c be a cycle of length not exceeding $4d$ (Figure 3). Then the set of the levels of the vertices on c form an interval $[a,b]$ of integers and let v be a vertex on level $l = \lceil (a+b)/2 \rceil$. Then each vertex on c is at distance at most d from v. Hence, if the level l is between levels $(i+1)d$ and $(i+2)d-1$, then all vertices from c will belong to G_i. Thus requirement (iii) is also satisfied. □

Next we show how to compute the length of a shortest cycle efficiently for weighted graphs with small separators using divide-and-conquer.

Algorithm SMALL_SEPARATOR

{Finds the girth of a graph G with nonnegative costs $cost(\cdot)$ on its edges if G has $O(\tau$ separator and $degree(G) = O(1)$}

1. Construct a separator S of G of size $O(\tau)$ dividing the graph into components of no more than $n/2$ vertices each, where n denotes the number of vertices of G.
2. For each component K of the partition of G find the length of the shortest cycle in K by this algorithm recursively.
3. For each edge e incident to a vertex from S find the length $c(e)$ of the shortest cycle in G containing e as follows.
   (a) Compute the distance $c'(e)$ between the endpoints of e in $G - e$ by running the linear time single-source shortest paths algorithm from [17] on $G - e$ with source any of the endpoints of e.
   (b) Assign $c(e) := c'(e) + cost(e)$.
4. Return the minimum length of a cycle found in Step 2 or Step 3.

Lemma 3. *Algorithm* SMALL_SEPARATOR *computes the girth of G in $O(\tau n \log n)$ time, assuming the tree-width of G is no more than τ.*

Proof: *Correctness.* Let c be a shortest cycle in G. If c contains a vertex of S, then its length will be computed in some iteration of Step 3. Otherwise c will be contained in some component of $G - S$ and its length will be computed in Step 2.
Time complexity. The time needed for Step 1 is $O(n)$, where n is the number of vertices of G [18]. Since each vertex of S has degree $O(1)$, then the number of iterations of Step 3 is $O(|S|) = O(\tau)$. The time for each iteration is dominated by the time for running the shortest paths algorithm in Step 3(a), which is $O(n)$. This implies $O(\tau n)$ time bound for Step 3. Thus the total time $T(n)$ for Algorithm SMALL_SEPARATOR satisfies the recurrence

$$T(1) = O(1),$$
$$T(n) \leq 2T(n/2) + O(\tau n), \quad \text{for } n > 1,$$

whose solution is $T(n) = O(\tau n \log n)$. □

In a preprocessing step, we transform G into a graph G' of maximum vertex degree three, where any edge e of G' has a cost $cost(e)$ equal to 1 or 0 depending on whether e corresponds to an edge of the original graph or not (see Figure 1). Then we apply to G' the following algorithm which is based on Algorithm SMALL_SEPARATOR and Lemma 2.

Algorithm SMALL_GIRTH

{Finds the girth of an n-vertex planar graph G assuming $girth(G) \leq \gamma$}

1. Apply Lemma 2 on G with $l = \lceil \gamma \log n \rceil$. Let $G_1, \ldots, G_s$ be the resulting subgraph cover of G.
2. For each G_i, $i = 1, \ldots, s$, find the length c_i of the shortest cycle in G_i ($c_i = \infty$ if G_i is acyclic) by applying Algorithm SMALL_SEPARATOR.
3. Return $girth(G) := \min\{c_i \mid i = 1, \ldots, s\}$.

Lemma 4. *Algorithm* SMALL_GIRTH *finds the girth of a planar graph G in $O(\gamma n \log^2 n)$ time, assuming $girth(G) \leq \gamma$*

Proof: The correctness follows from Lemma 2 and the correctness of Algorithm SMALL_SEPARATOR. The time for Step 1, and Step 3 is $O(n)$. By Lemma 3, the time for computing c_i in Step 2 is $O(\gamma \log n \, n_i \log n_i)$, where n_i is the number of the vertices of G_i, since by Lemma 2 each graph G_i has a separator of size $O(l)$, where $l = \gamma \log n$. Thus the total time for Step 2 is

$$\sum_{i=1}^{s} O(\gamma \log n \, n_i \log n_i) \leq O(\gamma \log^2 n) \sum_{i=1}^{s} n_i$$

$$= O(\gamma \log^2 n) O(n) = O(\gamma n \log^2 n).$$

□

5 The general case

In the case of an arbitrary planar input graph G we find the girth of G simply by applying one of the algorithms from the previous two sections depending on the maximum face size of a given embedding of G.

Algorithm FIND_GIRTH

{Finds the girth of an arbitrary n-vertex planar graph G}

1. If G is not biconnected, then compute its biconnected components and apply the remaining steps of this algorithm on any of the biconnected components (since each cycle in a graph belongs to exactly one of its biconnected components).
2. Transform G into a weighted planar graph G' with $n' = O(n)$ vertices and of a maximum vertex degree three (as discussed in Section 3 and illustrated on Figure 1).
3. Embed G' in the plane and find the maximum face size h of the embedding.

4. If $h = O(n^{1/4}/\log n)$, then compute the girth of G by applying on G' Algorithm SMALL_GIRTH.
5. Else (if $h = \Omega(n^{1/4}/\log n)$) compute the girth by applying on G' Algorithm LARGE_GIRTH.

Theorem 3. *Algorithm* FIND_GIRTH *finds the girth of an n-vertex planar graph in $O(n^{5/4}\log n)$ time.*

Proof: The correctness of the algorithm follows from Lemma 4 and Lemma 1.

If $h = O(n^{1/4}/\log n)$, then the time of the algorithm will be determined by the time for running Algorithm SMALL_GIRTH. By Lemma 4, that time is $O(\gamma n \log^2 n)$, where $\gamma = girth(G)$. Since $\gamma \leq h = O(n^{1/4}\log n)$, then the time of the algorithm in this case is $O((n^{1/4}/\log n)\, n \log^2 n) = O(n^{5/4}\log n)$.

If $h = \Omega(n^{1/4}/\log n)$, then by Lemma 1 the time for running Algorithm FIND_GIRTH will be $O(n^{3/2}/\gamma) = O(n^{3/2}/\,(n^{1/4}/\log n)\,) = O(n^{5/4}\log n)$. □

6 Related problems

The technique discussed here can be used to solve other problems related to shortest cycles in graphs. Algorithm FIND_GIRTH can be used to find a shortest non-facial cycle of a planar graph in $O(n^{5/4}\log n)$ time. Furthermore, using graph separation and divide-and-conquer, we can compute the girth of a directed graph in $O(n^{3/2})$ time. Finally, making use of the $O(\sqrt{gn})$ separator theorem for graphs of genus $g > 0$ [8, 14, 10], one can construct efficient algorithms for the above problems for graphs of genus $g = o(n)$. We will describe algorithms for solving the above problems in the full version of the paper.

This paper leaves several open problems, including the following:

1. Construct an $o(n^{3/2})$ time algorithm for computing the girth of a directed planar graph.
2. Develop efficient algorithms for finding shortest cycles in graphs with arbitrary nonnegative costs (lengths) on edges.
3. Construct $o(nm)$ algorithms for finding the girth of general graphs.

It will be also interesting to implement the algorithm from this paper and test its practicality.

References

1. Alon, Yuster and Zwick. Color-coding. *Journal of the ACM*, 42:844-856, 1995.
2. Alon, Yuster and Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209-223, 1997.
3. Brenda S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, 41(1):153–180, January 1994.

4. Béla Bollobás. Chromatic number, girth and maximal degree. *Discrete Mathematics*, 24:311–314, 1978.
5. R. J. Cook. Chromatic number and girth. *Periodica Mathematica Hungarica*, 6(1):103–107, 1975.
6. Reinhard Diestel. *Graph Theory*. Springer-Verlag, Berlin, Heidelberg, New York, Tokio, 1997.
7. H. Djidjev, G. Pantziou, and C. Zaroliagis. Improved algorithms for dynamic shortest paths. *Algorithmica*, 1999. To appear.
8. Hristo N. Djidjev. A separator theorem. *Compt. rend. Acad. bulg. Sci.*, 34:643–645, 1981.
9. Hristo N. Djidjev. On the problem of partitioning planar graphs. *SIAM Journal on Algebraic and Discrete Methods*, 3:229–240, 1982.
10. Hristo N. Djidjev. A linear algorithm for partitioning graphs of fixed genus. *Serdica*, 11:329–341, 1985.
11. David Eppstein. Subgraph isomorphism for planar graphs and related problems. In *Proc. 6th Symp. Discrete Algorithms*, pages 632–640. Assoc. Comput. Mach. and Soc. Industrial & Applied Math., 1995.
12. Paul Erdös. Graph theory and probability. *Canad. J. Math.*, 11:34–38, 1959.
13. G.N. Frederickson. Planar graph decomposition and all pairs shortest paths. *Journal of the ACM*, 38:162–204, 1991.
14. John R. Gilbert, Joan P. Hutchinson, and Robert E. Tarjan. A separator theorem for graphs of bounded genus. *J. Algorithms*, 5:391–407, 1984.
15. F. Harary. *Graph Theory*. Addison-Wesley, Reading, Massachusetts, 1969.
16. A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM J. Computing*, 7:413–423, 1978.
17. P. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest-path algorithms for planar graphs. In *26th ACM Symp. Theory of Computing*, pages 27–37, 1994.
18. Richard J. Lipton and Robert E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math*, 36:177–189, 1979.
19. L. Lovasz. On chromatic number of finite set systems. *Acta Math. Acad. Sci. Hun.*, 19:59–67, 1968.
20. Wolfgang Mader. Topological subgraphs in graphs of large girth. *Combinatorica*, 18:405, 1998.
21. B. Monien. The complexity of determining a shortest cycle of even length. *Computing*, 31:355–369, 1983.
22. Carsten Thomassen. Paths, circuits and subdivisions. In *Selected Topics in Graph Theory, ed. Lowell W. Beineke and Robin J. Wilson, Academic Press*, volume 3. 1988.
23. Vazirani and Yannakakis. Pfaffian orientations, 0-1 permanents, and even cycles in directed graphs. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 25, 1989.

Lower Bounds Are Not Easier over the Reals: Inside PH

Hervé Fournier and Pascal Koiran

Laboratoire de l'Informatique du Parallélisme
Ecole Normale Supérieure de Lyon
46, allée d'Italie
69364 Lyon Cedex 07, France
[Herve.Fournier,Pascal.Koiran]@ens-lyon.fr

Abstract. We prove that all NP problems over the reals with addition and order can be solved in polynomial time with the help of a boolean NP oracle. As a consequence, the "P = NP?" question over the reals with addition and order is equivalent to the classical question. For the reals with addition and equality only, the situation is quite different since P is known to be different from NP. Nevertheless, we prove similar transfer theorems for the polynomial hierarchy.

1 Introduction

Just as in discrete complexity theory, the main goal of algebraic complexity theory is to prove superpolynomial lower bounds for certain "natural" problems. In several algebraic settings this goal has not been achieved yet. For instance, it is not known whether the resultant of two sparse univariate polynomials can be computed by straight-line programs of polynomial length (see [12] for a motivation); the problem "VP = VNP?" in Valiant's model of computation [13,14] is still open; and the same is true of the "P = NP?" problem in the most interesting versions of the Blum-Shub-Smale model. It is not always clear whether these algebraic questions are easier than the well-known open questions from discrete complexity theory. Indeed, it was shown in [3] that problems such as Knapsack can be solved in polynomial time on a real (multiplication-free) Turing machine under the hypothesis P = PSPACE. Therefore a superpolynomial lower bound (on the circuit size, or on the running time of a real Turing machine) for Knapsack would imply a separation of P from PSPACE. In this paper we investigate similar questions for smaller complexity classes. Our main result is the following transfer theorem.

Theorem 1. $\mathrm{P}^0_{\mathbb{R}_{ovs}} = \mathrm{NP}^0_{\mathbb{R}_{ovs}}$ *if and only if* $\mathrm{P} = \mathrm{NP}$.

This implies for instance that Knapsack is in $\mathrm{P}^0_{\mathbb{R}_{ovs}}$ under the hypothesis P = NP, which is a weaker hypothesis than P = PSPACE. Here $\mathrm{P}^0_{\mathbb{R}_{ovs}}$ stands for the class of decision problems that can be solved in polynomial time by parameter-free real Turing machines over the structure $\mathbb{R}_{ovs}$ (i.e., the only legal operations are

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 832–843, 2000.
© Springer-Verlag Berlin Heidelberg 2000

$+$, $-$ and $<$). The main result of [3] was that $\mathrm{P}^0_{\mathbb{R}_{ovs}} = \mathrm{PAR}^0_{\mathbb{R}_{ovs}}$ is equivalent to P = PSPACE (PAR stands for "parallel polynomial time"). The complexity theory of $\mathbb{R}_{ovs}$ has been studied in [5,2]. More background on computation over the reals and other algebraic structures can be found in the textbooks [1,9].

In this paper, real Turing machines will be parameter-free unless stated otherwise. Results for machines with parameters follow from those for parameter-free machines. For instance:

Corollary 1. $\mathrm{P}_{\mathbb{R}_{ovs}} = \mathrm{NP}_{\mathbb{R}_{ovs}}$ *if and only if* $\mathrm{P/poly} = \mathrm{NP/poly}$.

Our proof of Theorem 1 relies on Meyer auf der Heide's construction of linear decision trees for point location in arrangements of hyperplanes [7,8]. Roughly speaking, we show in Theorem 2 that his construction can be made uniform if a boolean NP oracle is available. This result is established in section 2, and complexity-theoretic consequences are drawn in section 3 (as a byproduct, we obtain the unexpected result that problems such as the real Traveling Salesman Problem or Knapsack are $\mathrm{NP}_{\mathbb{R}_{ovs}}$-complete for Turing machine reductions). Here Theorem 3 is a key result: problems in $\mathrm{NP}^0_{\mathbb{R}_{ovs}}$ can be solved in polynomial time with the help of a boolean NP oracle. Theorem 1, its corollary, and the completeness results just mentioned then follow immediately.

In the order-free structure $\mathbb{R}_{vs}$ (where addition, subtraction and equality tests are the only operations allowed) the situation is quite different than in $\mathbb{R}_{ovs}$ since it is possible to prove the unconditional result $\mathrm{P}_{\mathbb{R}_{vs}} \neq \mathrm{NP}_{\mathbb{R}_{vs}}$, as shown originally by Meer [6]. It would be interesting to obtain other separation results in this structure. Unfortunately, for several questions (such as the collapse of the polynomial hierarchy $\mathrm{PH}_{\mathbb{R}_{vs}}$ and the separation of $\mathrm{PH}_{\mathbb{R}_{vs}}$ from $\mathrm{PAR}_{\mathbb{R}_{vs}}$) this turns out to be impossible with current techniques: the transfer theorems in section 4 show that these questions are as hard as outstanding open problems from discrete complexity theory.

2 Point Location in an Arrangement of Hyperplanes

We first recall some terminology regarding arrangements of hyperplanes. Let $H = \{h_1, \ldots, h_m\}$ be a set of hyperplanes in $\mathbb{R}^n$. We denote by h_i^+ and h_i^- the two open half-spaces defined by h_i. For a point $x \in \mathbb{R}^n$, we set $z_i(x) = 0$ if $x \in h_i$ and $z_i(x) = 1$ (respectively, -1) if $x \in h_i^+$ (respectively, $x \in h_i^-$). We define $\varphi(x) = (z_1(x), \ldots, z_m(x))$. The faces of the arrangement $\mathcal{A}(H)$ are by definition the classes of the equivalence relation $x \sim y \Leftrightarrow \varphi(x) = \varphi(y)$ on $\mathbb{R}^n$. The dimension of a face is the dimension of its affine closure; a face of dimension 0 is called a vertex, and a face of dimension n a cell.

We can now state the problem. Let $\mathcal{H}_n$ be the set of all hyperplanes defined by equations with integer coefficients in $\{-2^{t(n)}, \ldots, 2^{t(n)}\}$, where t is some fixed polynomial. We say that an algorithm solves the location problem (for the family of arrangements $H_t = (\mathcal{H}_n)_{n\in\mathbb{N}}$) if, on an input point $(x_1, \ldots, x_n) \in \mathbb{R}^n$, it

computes a system

$$\mathcal{S} = \begin{cases} f_i(y) < 0 & i = 1, \ldots, p \\ g_j(y) = 0 & j = 1, \ldots, r \end{cases}$$

made up of $p + r = n^{O(1)}$ affine (in)equations with integer polynomial-size coefficients such that the set of points $P_{\mathcal{S}}$ of $\mathbb{R}^n$ satisfying $\mathcal{S}$ is included in a face of $\mathcal{A}(\mathcal{H}_n)$, and $x \in P_{\mathcal{S}}$.

Theorem 2. *The location problem for H_t is in $\mathrm{FP}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$ for any polynomial $t \in \mathbb{Z}[X]$. This means that a system locating the input point can be computed in polynomial time by a Turing machine over $\mathbb{R}_{ovs}$ using a boolean* NP *oracle.*

Defining formally the model of "real Turing machine with a boolean oracle" used in this theorem would be tedious but completely straightforward. The idea is that such a Turing machine can only use the instructions "write-0" or "write-1" to write on its oracle tape. This ensures that the oracle query is a word of $\{0, 1\}^*$, despite the fact that the other tapes of the Turing machine may contain arbitrary real numbers.

Before proving the theorem we have to make an observation about (parameter-free) algorithms over the structure $\mathbb{R}_{ovs}$. By running such an algorithm on the formal input $(X_1, \ldots, X_n)$ and taking all possible paths into account, it is clear that each test is of the form $\sum_{i=1}^{n} a_i X_i + a_{n+1} > 0$ ($a_i \in \mathbb{Z}$). Thus, to a test on an input of length n corresponds an oriented hyperplane of $\mathbb{R}^n$ (having an equation with integer coefficients). This allows us to define a notion of size for the coefficients of tests:

Remark 1 *All tests performed by a program running in time $q(n)$ have coefficients bounded by $2^{q(n)}$.*

Let t be a polynomial in $\mathbb{Z}[X]$, and $\mathcal{L}_n \subseteq \mathbb{R}^n$ the union of the hyperplanes in the arrangement $\mathcal{H}_n$ defined in section 2. Before solving the point location problem for $\mathcal{H}_n$ we will describe an algorithm for recognizing $\mathcal{L}_n$. The union of the $\mathcal{L}_n$ for $n \geq 1$ is a language of $\mathbb{R}^\infty$ denoted L_t.

Proposition 1. *For any polynomial t, L_t is in $\mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$.*

The remainder of this section is devoted to the proof of Proposition 1 and Theorem 2. The algorithms are based on a construction of Meyer auf der Heide [7,8], who has described families of polynomial depth linear decision trees deciding unions of hyperplanes.[1] We shall build a uniform machine with an oracle in NP performing the same computation. The proof of Proposition 1 is in three steps: in section 2.1 we describe an algorithm for deciding a union of hyperplanes on $[-1, 1]^n$. The size of the tests in this algorithm is analyzed in section 2.2, and the algorithm for deciding a union of hyperplanes in $\mathbb{R}^n$ is then obtained in section 2.3. Finally, this algorithm is turned into a point location algorithm in section 2.4.

[1] As mentioned in the introduction, he has also described families of linear decision trees solving the whole location problem.

2.1 Deciding a Union of Hyperplanes on a Cube

We now describe a recursive method for deciding $\mathcal{L}_n = \bigcup_{h \in \mathcal{H}_n} h$ on $[-1,1]^n$.

Lemma 1. *There is a* $\mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$ *algorithm which, for any input* $x \in \mathbb{R}^n$*, decides whether* $x \in \mathcal{L}_n \cap [-1,1]^n$.

For the complete proof that the algorithm described below really is $\mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$ we need polynomial size bounds on the coefficients of tests. These bounds are established in section 2.2.

Given a point $y \in \mathbb{R}^n$ and a set $A \subseteq \mathbb{R}^n$, we denote by $\mathrm{Aff}(y, A)$ the affine closure of $\{y\} \cup A$, and by $P(y, A) = \{\lambda y + (1-\lambda)x, x \in A, \lambda < 1\}$ the pyramid of top y and base A. Recursion on the dimension n of the cube is made possible by the following lemma.

Lemma 2 (Meyer auf der Heide). *Let* $S = \{h_1, \ldots, h_p\}$ *be a set of hyperplanes in* $\mathbb{R}^n$ *such that the intersection* $I = \bigcap_{i=1}^p h_i$ *is nonempty. Let* A *be a polytope on a hyperplane* h_0 *which does not contain* I*, and let* s *be a point in* $I \setminus h_0$*. If a program decides* $L' = \bigcup_{i=1}^p h_i \cap h_0$ *on* A*, then the program obtained by replacing each test* h' *by* $\mathrm{Aff}(s, h')$ *(with the appropriate sign) decides* $L = \bigcup_{i=1}^p h_i$ *on* $P(s, A)$.

Let $x \in P(s, A)$. The previous lemma is clear when we notice the equivalence

$$x \in h_1 \cup \ldots \cup h_p \Leftrightarrow (sx) \cap h_0 \in (h_1 \cap h_0) \cup \ldots \cup (h_p \cap h_0)\ .$$

Now we need a definition. A number $r > 0$ is a *coarseness* of a set of hyperplanes $h_1, \ldots, h_k$ of $\mathbb{R}^n$ if, for any ball B of radius r, either $\{h_i, h_i \cap B \neq \emptyset\} = \emptyset$ or $\bigcap_{h_i \cap B \neq \emptyset} h_i \neq \emptyset$. Let r_n be a coarseness of $\mathcal{H}_n$. In [8] it is shown that one can take $1/r_n = n^{n^2} 2^{2n^2 t(n) + O(n^2)}$.

Here is how the algorithm works.
If $n = 1$ we can decide whether $x \in \mathcal{L}_n$ by binary search (no NP oracle is needed). We now assume that $n > 1$, and set $H^0_n = \mathcal{H}_n$.
▷ Step 1.
We subdivide the cube $C^1_n = [-1,1]^n$ in little cubes with radius smaller than r_n (i.e., with edge length smaller than $2r_n/\sqrt{n}$). By binary search on each coordinate, we find a little cube c^1_n such that $x \in c^1_n$. Let us call $H^1_n = \{h \in H^0_n,\ h \cap c^1_n \neq \emptyset\}$. There are two cases :

(i) $H^1_n = \emptyset$.
(ii) Otherwise, $\bigcap_{h \in H^1_n} h \neq \emptyset$ by definition of coarseness.

We can check with a boolean NP algorithm whether (i) holds, and reject the input if this is the case. If it turns out that we are in case (ii) we compute in polynomial time with the help of a boolean NP oracle a point s^1_n in $I^1 = \bigcap_{h \in H^1_n} h$. In order to do this we will in fact compute a strictly decreasing sequence $E_1, \ldots, E_j$ of affine subspaces such that E_1 is an element of H^1_n and

$E_j = I^1$ (note that $j \leq n$). Since the condition $H_n^1 \neq \emptyset$ is in NP, we can compute E_1 by prefix search with an NP oracle. E_{i+1} can be computed from E_i as follows. If it turns out that $E_i \subseteq h$ for all $h \in H_n^1$ we can halt with $I^1 = E_i$. This condition can again be checked with a boolean NP oracle. Otherwise there exists $h \in H_n^1$ such that $\dim(h \cap E_i) = \dim E_i - 1$ (since we are in case (ii) the case $h \cap E_i = \emptyset$ is not possible). Such an h can be determined by prefix search, and we set $E_{i+1} = E_i \cap h$. Finally, when $E_j = I^1$ has been determined we can easily compute a point s_n^1 on this affine space (e.g. by fixing some coordinates 0). If $x = s_n^1$ we accept the input. Otherwise we repeat the same procedure in dimension $n-1$ as explained below.

First we determine a face f_n^1 of the cube C_n^1 such that x is in $P(s_n^1, f_n^1)$, the pyramid of top s_n^1 and base f_n^1. Let g_n^1 be the affine closure of f_n^1 (the equation of g_n^1 is of the form $x_i = \pm 1$). Then Lemma 2 applies, so it remains to solve a $(n-1)$-dimensional problem: decide whether the point $(s_n^1 x) \cap f_n^1$ lies on $\bigcup_{h \in H_n^1} (g_n^1 \cap h)$ on the $(n-1)$-dimensional cube $[-1,1]^{n-1}$ of g_n^1. An important point is that if r is a coarseness of a set $\{h, h_1, \ldots, h_p\}$ of hyperplanes, then r is a coarseness of $h \cap h_1, \ldots, h \cap h_p$ on h. Since the hyperplane which plays the role of h (namely g_n^1) is an element of $\mathcal{H}_n$, this will allow us to subdivide the $(n-1)$-dimensional cube with the same r_n (and this remains true at further steps of the recursion).

▷ Step k $(1 < k \leq n)$.
At this step, we work on the cube $C_n^k = [-1,1]^{n-k+1}$ of the affine space $\{x_{i_1} = \varepsilon_1, \ldots, x_{i_{k-1}} = \varepsilon_{k-1}\}$ with a projected point x^k (the values of $\varepsilon_j \in \{-1,1\}$ and of the i_j depend on which face of C_n^j was chosen as base of the pyramid at step j). We subdivide C_n^k in smaller cubes, and then locate x^k in a little cube c_n^k of C_n^k. Note that the coordinates of x^k need not be computed explicitly. Instead, a test h' on x^k is done by performing the test $\mathrm{Aff}(s_n^1, (\mathrm{Aff}(s_n^2, \mathrm{Aff}(\ldots, \mathrm{Aff}(s_n^{k-1}, h')\ldots)$ on x. Let H_n^k be the subset of hyperplanes of $\mathcal{H}_n$ that intersect all little cubes $c_n^1, \ldots, c_n^k$ found up to step k. We know that if x lies on an hyperplane of $\mathcal{H}_n$, this point must in fact lie on an hyperplane of H_n^k. If $H_n^k = \emptyset$ we reject x as in step (i). Otherwise we compute a point $s_n^k \in \bigcap_{h \in H_n^k} h$ as in step (ii). If $k = n$ we accept x if $x = s_n^k$, and reject otherwise. If $k < n$ we determine i_k and ε_k, and go one step further into the recursion.

2.2 Coefficients of Tests in the Location Algorithm

What is the running time of the procedure described in section 2.1? As $r_n = 2^{n^{O(1)}}$, locating a point in a small cube by binary search always takes polynomial time. Moreover, it is clear that the number of operations performed (over $\mathbb{Z}$) to compute the coefficients of tests is polynomially bounded. To make sure that the algorithm is really $\mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$, it just remains to check that the coefficients of these tests are of polynomial size. Details can be found in [4].

2.3 Deciding a Union of Hyperplanes on $\mathbb{R}^n$

Following Meyer auf der Heide, we can extend the method described previously to decide a union of hyperplanes on $\mathbb{R}^n$. This must be done without multiplication, and we have to keep the coefficients small. Details can be found in [4]. This ends the proof of proposition 1.

2.4 Proof of the Location Theorem

We are now ready for the proof of Theorem 2. Let $x = (x_1, \ldots, x_n)$ be the point to be located. We use a perturbation method to turn the algorithm of section 2.3 into a point location algorithm. Set $\tilde{x} = (x_1 + \varepsilon_1, \ldots, x_n + \varepsilon_n)$ where $\epsilon_1, \ldots, \epsilon_n$ are infinitely small, positive, and $\varepsilon_1 \ll \epsilon_2 \ll \cdots \ll \varepsilon_n$. Now we run on input $\tilde{x}$ the algorithm of Proposition 1 for deciding $\mathcal{L}_n = \bigcup_{h \in \mathcal{H}_n} h$. Of course we know in advance that $\tilde{x}$ will be rejected since this input does not lie on any hyperplane with integer coefficients; the point is that the collection of all tests performed on $\tilde{x}$ during the execution of this algorithm is a system which locates $\tilde{x}$ in $\mathcal{A}(\mathcal{H}_n)$. Let $\tilde{S} = \{f_1(y) < 0, \ldots, f_q(y) < 0\}$ be this system. Then for each i we test whether $f_i(x) < 0$ or $f_i(x) = 0$: this yields a new system S, which locates x in $\mathcal{A}(\mathcal{H}_n)$. Moreover the size conditions are fulfilled: S is made of $n^{O(1)}$ affine (in)equations with integers coefficients of size $n^{O(1)}$.

3 Transfer Theorems for $\mathbb{R}_{ovs}$

We first recall some notations and results on the structure $\mathbb{R}_{ovs} = (\mathbb{R}, +, -, <)$ of the reals with addition and order. A real language (or *real problem*) is a subset of $\mathbb{R}^\infty = \coprod_{n \in \mathbb{N}} \mathbb{R}^n$. The boolean part $\mathrm{BP}(L)$ of a language $L \subseteq \mathbb{R}^\infty$ is by definition $L \cap \{0,1\}^\infty$. For a class $\mathcal{C}$ of real languages, $\mathrm{BP}(\mathcal{C}) = \{\mathrm{BP}(L),\ L \in \mathcal{C}\}$.

Fact 1 $\mathrm{BP}(\mathrm{P}^0_{\mathbb{R}_{ovs}}) = \mathrm{P}$, $\mathrm{BP}(\mathrm{NP}^0_{\mathbb{R}_{ovs}}) = \mathrm{NP}$, $\mathrm{BP}(\mathrm{P}_{\mathbb{R}_{ovs}}) = \mathrm{P/poly}$ *and* $\mathrm{BP}(\mathrm{NP}_{\mathbb{R}_{ovs}}) = \mathrm{NP/poly}$.

We recall that $\mathrm{NDP}^0_{\mathbb{R}_{ovs}}$ is a "digital" version of $\mathrm{NP}^0_{\mathbb{R}_{ovs}}$ where certificates are required to be boolean. More precisely, a problem $A \subseteq \mathbb{R}^\infty$ is in $\mathrm{NDP}^0_{\mathbb{R}_{ovs}}$ if there exists $B \in \mathrm{P}^0_{\mathbb{R}_{ovs}}$ and a polynomial p such that

$$x = (x_1, \ldots, x_n) \in A \Leftrightarrow \exists z \in \{0,1\}^{p(n)} \langle x, z \rangle \in B\ .$$

Fact 2 $\mathrm{NP}^0_{\mathbb{R}_{ovs}} = \mathrm{NDP}^0_{\mathbb{R}_{ovs}}$.

The proofs of these results can be found in [5] and [2]. We can now state and prove a key transfer theorem.

Theorem 3. $\mathrm{NP}^0_{\mathbb{R}_{ovs}} \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$.

Proof. Let $L \in \mathrm{NP}^0_{\mathbb{R}_{ovs}}$. By Fact 2, there exists $A \in \mathrm{P}^0_{\mathbb{R}_{ovs}}$ and a polynomial r such that

$$(x_1, \ldots, x_n) \in L \Leftrightarrow \exists z \in \{0,1\}^{r(n)} \langle x, z\rangle \in A.$$

For any $x \in \mathbb{R}^n$ and $z \in \{0,1\}^{r(n)}$, the condition $\langle x, z\rangle \in A$ can be checked in polynomial time $t(n)$ by a Turing machine T over $\mathbb{R}_{ovs}$, and the polynomial t depends only on A. Let $\mathcal{H}_n$ be the set of all hyperplanes of $\mathbb{R}^n$ with coefficients in $\{-2^{t(n)}, \ldots, 2^{t(n)}\}$. For any $z \in \{0,1\}^{r(n)}$, if we run $A\langle ., z\rangle$ on the formal input $(X_1, \ldots, X_n)$, each test is of the form $\sum_{i=1}^n a_i X_i + a_{n+1} > 0$, with a_i in $\{-2^{t(n)}, \ldots, 2^{t(n)}\}$. As a consequence, $L \cap \mathbb{R}^n$ is a union of faces of $\mathcal{A}(\mathcal{H}_n)$. The $\mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$ algorithm deciding L works in two steps.

First, the input $x = (x_1, \ldots, x_n)$ is located in $\mathcal{A}(\mathcal{H}_n)$. By Theorem 2, this can be done in $\mathrm{FP}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$. The output is a system $\mathcal{S}$ of $n^{O(1)}$ (in)equations of the form $h_i(x) < 0$ or $h_i(x) = 0$ such that the h_i's have polynomial size coefficients and the set $P_{\mathcal{S}}$ of the points of $\mathbb{R}^n$ satisfying $\mathcal{S}$ is included in a face of $\mathcal{A}(\mathcal{H}_n)$.

Then it remains to check whether $P_{\mathcal{S}}$ is included in L or in its complement. This can be done by a standard NP algorithm: we guess a certificate $z \in \{0,1\}^{r(n)}$ and an accepting computation path of T. The set of inputs of $\mathbb{R}^n$ which given z follow this computation path is a polyhedron P defined by a system of linear inequalities of polynomial size. We accept x if $P \cap P_{\mathcal{S}} \neq \emptyset$. This linear programming problem can be solved in polynomial time. (Another method consists in guessing a rational point $q \in P_{\mathcal{S}}$ with small coordinates – such a point exists, see [5] – and then running T on $\langle q, z\rangle$).

Remark 2 *Other inclusions can be obtained with this method. The point location algorithm described above can be used for any real language L in a complexity class $\mathcal{C}^0_{\mathbb{R}_{ovs}} \subseteq \mathrm{PAR}^0_{\mathbb{R}_{ovs}}$. Then it remains to check whether the resulting polyhedron $P_{\mathcal{S}}$ is included in L or in its complement. If $\mathcal{C}$ is "reasonable", this will be feasible in $\mathcal{C}_{\mathbb{Z}_2}$. For example, we have $\mathrm{PP}^0_{\mathbb{R}_{ovs}} \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{PP})$, $(\Sigma^k_{\mathbb{R}_{ovs}})^0 \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(\Sigma^k)$ for $k \in \mathbb{N}$ and $\mathrm{PAR}^0_{\mathbb{R}_{ovs}} \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{PSPACE})$. Of course we have $(\Pi^k_{\mathbb{R}_{ovs}})^0 \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(\Pi^k)$ too. For* BPP*, we only obtain $\mathrm{BPP}^0_{\mathbb{R}_{ovs}} \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP} \oplus \mathrm{BPP})$ where $\oplus$ is the join operation.*

The results stated in the introduction (Theorem 1 and its corollary) are direct consequences of Theorem 3.

Proof (of Theorem 1). If $\mathrm{P}^0_{\mathbb{R}_{ovs}} = \mathrm{NP}^0_{\mathbb{R}_{ovs}}$, $\mathrm{P} = \mathrm{NP}$ by Fact 1. The converse follows immediately from Theorem 3.

Proof (of Corollary 1). If $\mathrm{P}_{\mathbb{R}_{ovs}} = \mathrm{NP}_{\mathbb{R}_{ovs}}$, $\mathrm{P/poly} = \mathrm{NP/poly}$ by Fact 1. For the converse, consider a problem A in $\mathrm{NP}_{\mathbb{R}_{ovs}}$. There exists $B \in \mathrm{NP}^0_{\mathbb{R}_{ovs}}$ and parameters $\alpha_1, \ldots, \alpha_p$ such that $(x_1, \ldots, x_n) \in A \Leftrightarrow (x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_p) \in B$. By Theorem 3 $B \in \mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$, hence $B \in \mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{P/poly})$ by the assumption $\mathrm{P/poly} = \mathrm{NP/poly}$. Encoding the advice function in an additional parameter yields $B \in \mathrm{P}_{\mathbb{R}_{ovs}}$, therefore $A \in \mathrm{P}_{\mathbb{R}_{ovs}}$ too.

We will now give a completeness result. For a real language $L \subseteq \mathbb{R}^\infty$, let us define the integer part of L as

$$\mathrm{IP}(L) = \bigcup_{n\in\mathbb{N}} \{\langle p_1, \ldots, p_n\rangle,\ (p_1, \ldots, p_n) \in L \text{ and } p_1, \ldots, p_n \in \mathbb{Z}\}.$$

For a real complexity class $\mathcal{C}$, we set $\mathrm{IP}(\mathcal{C}) = \{\mathrm{IP}(L),\ L \in \mathcal{C}\}$.

Lemma 3. *Let $A \subseteq \mathbb{R}^\infty$ be such that $\mathrm{IP}(A)$ is Turing NP-hard. Then A is Turing $\mathrm{NP}^0_{\mathbb{R}_{ovs}}$-hard.*

Proof. By Theorem 3, $\mathrm{NP}^0_{\mathbb{R}_{ovs}} \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP})$. As $\mathrm{IP}(A)$ is NP-hard, $\mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{NP}) \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{IP}(A))$, and of course $\mathrm{P}^0_{\mathbb{R}_{ovs}}(\mathrm{IP}(A)) \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(A)$. Here $\mathrm{IP}(A)$ is a boolean language used as a boolean oracle: as explained after Theorem 2, such an oracle only handles inputs made of 0's and 1's. We conclude that $\mathrm{NP}^0_{\mathbb{R}_{ovs}} \subseteq \mathrm{P}^0_{\mathbb{R}_{ovs}}(A)$.

Let us recall the definitions of two classical real languages. The real knapsack problem $\mathrm{Knapsack}_\mathbb{R}$ is defined by

$$\mathrm{Knapsack}_\mathbb{R} \cap \mathbb{R}^{n+1} = \{(x_1, \ldots, x_n, s),\ \exists u_1, \ldots, u_n \in \{0, 1\},\ \sum_{i=1}^{n} u_i x_i = s\}\ .$$

The traveling salesman problem $\mathrm{TSP}_\mathbb{R}$ is the set of pairs (A, d) where A is a distance matrix of $\{1, \ldots, n\}$ and $d \in \mathbb{R}^+$, such that there exists a Hamiltonian tour over $\{1, \ldots, n\}$ of length at most d. The final result of this section follows immediately from Lemma 3.

Proposition 2. $\mathrm{Knapsack}_\mathbb{R}$ *and* $\mathrm{TSP}_\mathbb{R}$ *are Turing* $\mathrm{NP}^0_{\mathbb{R}_{ovs}}$*-complete.*

4 The Polynomial Hierarchy over $\mathbb{R}_{vs}$

The result $\mathrm{P}_{\mathbb{R}_{vs}} \neq \mathrm{NP}_{\mathbb{R}_{vs}}$ was proved by Meer [6]. In this section we show that similar arguments can be used to separate the lowest levels of the polynomial hierarchy $\mathrm{PH}_{\mathbb{R}_{vs}}$. Separating the higher levels of the hierarchy is essentially "impossible" due to the transfer theorems established in section 4.2. These results rely on elementary observations on the structure of subsets of $\mathbb{R}^n$ definable in $\mathbb{R}_{vs}$ (see Lemma 5 and Lemma 6 in particular). In the following, these subsets will simply be called "definable sets". Remember that definable sets are definable without quantifiers since $\mathbb{R}_{vs}$ admits quantifier elimination. Consequently, a definable set is nothing but a boolean combination of hyperplanes.

As in the rest of the paper, we work with parameter-free machines unless stated otherwise. We point out in Remark 4 at the end of the paper that it is straightforward to generalize our transfer theorems to the case of machines with parameters.

We first recall the generic path method. Let M be a machine over $\mathbb{R}_{vs}$ stopping on all inputs, and L the language decided by M. Given $n \in \mathbb{N} \setminus \{0\}$, we set $L_n = L \cap \mathbb{R}^n$. The generic path in M for inputs of size n is the path obtained by

answering *no* to all tests of the form $''h(x) = 0\ ?''$ (unless $h = 0$, in which case we answer *yes*).

This definition is effective in the sense that the generic path can be followed by running M on the formal input $(X_1, \ldots, X_n)$. If M is parameter-free this computation can be carried out on a classical Turing machine. Moreover, if M works in time $t(n)$, it takes time $O(nt(n)^2)$ to apply the generic path method, and the tests that are performed by M along this path can be computed effectively. Let us call $\{h_1, \ldots, h_r\}$ these tests. We have $r \leq t(n)$, and these hyperplanes have the following property: if the inputs following the generic path are rejected, $L_n \subseteq h_1 \cup \ldots \cup h_r$; otherwise these inputs are accepted and $L_n^c \subseteq h_1 \cup \ldots \cup h_r$.

Note that the generic path method can be applied to an affine subspace $X \subseteq \mathbb{R}^n$ instead of $\mathbb{R}^n$, in which case we answer *yes* to a test $''h(x) = 0\ ?''$ if and only if $X \subseteq h$. Remember also that a definable subset A of $\mathbb{R}^n$ is dense on X iff it contains an open dense subset of X, and that this is equivalent to $\dim A \cap X = \dim X$. We summarize these observations in a lemma which will be used in section 4.2. In this lemma, Sys^n denotes the set of systems of affine equations in n variables with coefficients in $\mathbb{Z}$. For $S \in \mathrm{Sys}^n$, P_S denotes the affine subspace of $\mathbb{R}^n$ defined by S.

Lemma 4. *Let A be a language of $\mathbb{R}^\infty$ and $A^n = A \cap \mathbb{R}^n$. We denote by L^n the set of systems $S \in \mathrm{Sys}^n$ such that A^n is dense in P_S, and by L the language $\bigcup_{n \geq 1} L^n$. Assume that $A \in \mathcal{C}^0_{\mathbb{R}_{vs}}$, with $\mathcal{C} = \mathrm{PAR}$ or $\mathcal{C} = \Sigma^k$ for some $k \in \mathbb{N}$. Then $L \in \mathcal{C}_{\mathbb{Z}_2}$.*

Proof. Note that A^n is definable for any $A \in \mathcal{C}^0_{\mathbb{R}_{vs}}$ (this is in fact true for any recursive language of $\mathbb{R}^\infty_{vs}$). We can therefore apply the generic path method described above to decide wether A^n is dense in P_S. More precisely, consider first the case $A \in \mathrm{P}^0_{\mathbb{R}_{vs}}$. Given a test hyperplane h, we can decide in polynomial time whether $P_S \subseteq h$ by linear algebra (for instance, we precompute $d = \dim(P_S)$ and $d+1$ points $x_1, \ldots, x_{d+1}$ such that $P_S = \mathrm{Aff}(x_1, \ldots, x_{d+1})$; then we declare that $P_S \subseteq h$ if $x_i \in h$ for all $i = 1, \ldots, d+1$). The same remark applies to the case $\mathcal{C} = \mathrm{PAR}$ since test hyperplanes still have polynomial-size coefficients in this case. We conclude that L is in P if $A \in \mathrm{P}^0_{\mathbb{R}_{vs}}$, and L is in $\mathrm{PAR}_{\mathbb{Z}_2} = \mathrm{PSPACE}$ if $A \in \mathrm{PAR}^0_{\mathbb{R}_{vs}}$.

If $A \in (\Sigma^k_{\mathbb{R}_{vs}})^0$ for some $k \geq 1$ we use the equivalence between real and boolean alternation for $\mathbb{R}_{vs}$ [2]: there exists a polynomial p and $B \in \mathrm{P}^0_{\mathbb{R}_{vs}}$ such that for any $x \in \mathbb{R}^n$, $x \in A$ iff

$$Q_1 y_1 \in \{0,1\}^{p(n)} \cdots Q_k y_k \in \{0,1\}^{p(n)} \langle x, y_1, \ldots, y_k \rangle \in B$$

(the quantifiers Q_i alternate, starting with $Q_1 = \exists$). The set A^n is dense in P_S iff the statement

$$Q_1 y_1 \in \{0,1\}^{p(n)} \cdots Q_k y_k \in \{0,1\}^{p(n)} F_n(y_1, \ldots, y_k) \tag{1}$$

is true. Here $F_n(y_1, \ldots, y_k)$ stands for: "$\{x \in \mathbb{R}^n;\ \langle x, y_1, \ldots, y_k \rangle \in B\}$ is dense in P_S". Since $B \in \mathrm{P}^0_{\mathbb{R}_{vs}}$, we know that $F_n(y_1, \ldots, y_k)$ can be decided in polynomial time by the generic path method. Therefore (1) shows that $L \in \Sigma^k$.

Note that in the case $\mathcal{C} = \Sigma^k$ it is really necessary to go to boolean quantification before applying the generic path method (think for instance of the set of points $x \in \mathbb{R}$ defined by the formula $\exists y\ x = y$).

4.1 Separation of Lower Levels

It was pointed out in [11] that the Twenty Questions problem might be a plausible witness to the separation $\mathrm{P}_{\mathbb{C}} \neq \mathrm{NP}_{\mathbb{C}}$. Formally, this problem is defined as follows:

$$\mathrm{TQ} = \bigcup_{n \in \mathbb{N}} \{(x_1, \ldots, x_n) \in \mathbb{R}^n,\ x_1 \in \{0, \ldots, 2^n - 1\}\}.$$

Twenty Questions can be used to separate $\Sigma^1_{\mathbb{R}_{vs}}$ from $\Pi^1_{\mathbb{R}_{vs}}$.

Proposition 3. $(\Sigma^2_{\mathbb{R}_{vs}} \cap \Pi^2_{\mathbb{R}_{vs}}) - (\Sigma^1_{\mathbb{R}_{vs}} \cup \Pi^1_{\mathbb{R}_{vs}}) \neq \emptyset$.

Proof is omitted.

4.2 Transfer Theorems for the Polynomial Hierarchy

In this section we show that it will be considerably harder to separate the higher levels of $\mathrm{PH}_{\mathbb{R}_{vs}}$ than the lower levels. We begin with two lemmas. Lemma 5 is a remark on the structure of definable subsets of $\mathbb{R}^n$, and in Lemma 6 we build a generic $\Sigma^2_{\mathbb{R}_{vs}}$ formula deciding a definable set A with the help of the predicate $\dim S \cap A = \dim S$ (the variable S represents an affine subset of $\mathbb{R}^n$).

Lemma 5. *Any nonempty definable set $A \subseteq \mathbb{R}^n$ can be written as*

$$A = E_k \setminus (E_{k-1} \setminus (\ldots \setminus E_0))$$

where E_i is a finite union of affine subspaces, $E_{i-1} \subsetneq E_i$, $E_i = \overline{E_i \setminus E_{i-1}}$, and $k \leq \dim A$.

Proof. If $\dim A = 0$ the result is clearly true since A is a finite set of points. Assume by induction that the result is true for all definable sets of dimension at most $d-1$, and let A be a definable set of dimension d. The topological closure $\overline{A}$ of A is a finite union of affine subspaces. If $A = \overline{A}$ we set $k = 0$ and $E_k = A$. Otherwise, consider the definable set $A_1 = \overline{A} \setminus A$. Since $\dim A_1 \leq d-1$, for some $k \leq d$ one can write by induction hypothesis $A_1 = E_{k-1} \setminus (E_{k-2} \setminus (\cdots \setminus E_0))$ with E_i a finite union of affine subspaces, $E_{i-1} \subsetneq E_i$, $E_i = \overline{E_i \setminus E_{i-1}}$. Since $A = \overline{A} \setminus A_1$, we can take $E_k = \overline{A}$.

Lemma 6. *For any definable set $A \subseteq \mathbb{R}^n$ we have:*

$$\begin{aligned}
&(x_1, \ldots, x_n) \in A \Leftrightarrow \exists S_1 \forall S_2 \\
&\quad x \in S_1 \wedge \dim A \cap S_1 = \dim S_1 \\
&\quad \wedge (\dim A \cap S_1 \cap S_2 < \dim S_1 \cap S_2 \Rightarrow x \notin S_1 \cap S_2)
\end{aligned}$$

where S_1 et S_2 are affine subspaces of $\mathbb{R}^n$.

Proof. The result is clearly true if $A = \emptyset$. Otherwise, write $A = E_k \setminus (E_{k-1} \setminus (\ldots \setminus E_0))$ as in Lemma 5. Let $x \in A$ and $i_0 = \min\{i;\ i = k \bmod 2,\ x \in E_i\}$. Then $x \notin E_{i_0-1}$: if x belonged to E_{i_0-1}, since $x \in A$ there would exist $i < i_0$ such that $i = k \bmod 2$ and $x \in E_i$. This would be in contradiction with the minimality of i_0.

We first show the implication from left to right: let S_1 be a maximal affine subspace in E_{i_0} containing x. Since $x \in S_1$ and $x \notin E_{i_0-1}$ the strict inclusion $S_1 \cap E_{i_0-1} \subsetneq S_1$ holds. Hence $\dim S_1 \setminus E_{i_0-1} = \dim S_1$ and $\dim A \cap S_1 = \dim S_1$. At last, if $\dim A \cap S_1 \cap S_2 < \dim S_1 \cap S_2$, then $S_1 \cap S_2 \subseteq E_{i_0-1}$. Thus $x \notin S_1 \cap S_2$.

Conversely, assume now that x satisfies the formula for $S_1 = S$. Since $A \cap S$ is definable, by Lemma 5 we can write $A \cap S = E_k \setminus (E_{k-1} \setminus (\ldots \setminus E_0))$. Here $E_k = \overline{A \cap S} = S$ (the second equality follows from $\dim A \cap S = S$). E_{k-1} is a finite union of affine subspaces. For any subspace S_2 in this union we have $\dim A \cap S \cap S_2 < \dim S \cap S_2$, therefore $x \notin S \cap S_2$. This shows that $x \notin E_{k-1}$, hence $x \in A \cap S$.

Remark 3 *If the definable set A in the above lemma is a boolean combination of hyperplanes with coefficients in some subset $\mathcal{D} \subseteq \mathbb{R}$, then we can quantify only on affine subspaces defined by systems of affine equations with coefficients in $\mathcal{D}$.*

We can now state and prove our transfer theorems for $\mathbb{R}_{vs}$. Note that there is a two level shift in Theorem 4 and Theorem 5.

Theorem 4. $\mathrm{P} = \mathrm{PSPACE} \Rightarrow \mathrm{PAR}^0_{\mathbb{R}_{vs}} = (\Sigma^2_{\mathbb{R}_{vs}})^0 \cap (\Pi^2_{\mathbb{R}_{vs}})^0$.

Proof. Let us assume that $\mathrm{P} = \mathrm{PSPACE}$, and let $L \in \mathrm{PAR}^0_{\mathbb{R}_{vs}}$ be decided by a family of P-uniform circuits with depth $t(n)$. It is enough to show that $\mathrm{PAR}^0_{\mathbb{R}_{vs}} = (\Sigma^2_{\mathbb{R}_{vs}})^0$ since $\mathrm{PAR}^0_{\mathbb{R}_{vs}}$ is closed under complement. By Lemma 6, L is decided by the following $(\Sigma^2_{\mathbb{R}_{vs}})^0$ formula:

$$\begin{aligned}
&(x_1, \ldots, x_n) \in L \Leftrightarrow \exists \mathcal{S}_1 \forall \mathcal{S}_2 \\
&\ x \in P_{\mathcal{S}_1} \wedge \dim L^n \cap P_{\mathcal{S}_1} = \dim P_{\mathcal{S}_1} \\
&\ \wedge (\dim L^n \cap P_{\mathcal{S}_1 \cup \mathcal{S}_2} < \dim P_{\mathcal{S}_1 \cup \mathcal{S}_2} \Rightarrow x \notin P_{\mathcal{S}_1 \cup \mathcal{S}_2})
\end{aligned}$$

where $L^n = L \cap \mathbb{R}^n$, $\mathcal{S}_1$ and $\mathcal{S}_2$ are systems of at most n affine equations with coefficients in $\{-2^{t(n)}, \ldots, 2^{t(n)}\}$ (Remark 3), and $P_{\mathcal{S}}$ is the subspace of $\mathbb{R}^n$ defined by $\mathcal{S}$. By Lemma 4 the condition $\dim L_n \cap P_{\mathcal{S}} = \dim P_{\mathcal{S}}$ can be checked in PSPACE, and therefore in P by hypothesis.

Theorem 5. *For all $k \geq 0$:*

$$\mathrm{PH} = \Sigma^k \Rightarrow \mathrm{PH}^0_{\mathbb{R}_{vs}} = (\Sigma^{k+2}_{\mathbb{R}_{vs}})^0.$$

Proof. Consider a problem $L \in \mathrm{PH}^0_{\mathbb{R}_{vs}}$: we have $L \in (\Sigma^q_{\mathbb{R}_{vs}})^0$ for some $q \geq 0$. As in the proof of the previous theorem, we use the Σ^2 formula from Lemma 6. Since

$\mathrm{PH}^0_{\mathbb{R}_{vs}} \subseteq \mathrm{PAR}^0_{\mathbb{R}_{vs}}$, by Remark 3 we can still quantify on systems of equations with coefficients in $\{-2^{p(n)}, \ldots, 2^{p(n)}\}$, for some polynomial p. By Lemma 4 the condition $\dim P_S \cap L_n = \dim P_S$ can be checked in Σ^q, and thus in Σ^k by hypothesis. Putting the resulting formula in prenex form shows that $L \in (\Sigma^{k+2}_{\mathbb{R}_{vs}})^0$.

Our final transfer theorem is based on a slightly different technique.

Theorem 6. $\mathrm{P} = \mathrm{NP} \Rightarrow (\Sigma^1_{\mathbb{R}_{vs}})^0 \cap (\Pi^1_{\mathbb{R}_{vs}})^0 = \mathrm{P}^0_{\mathbb{R}_{vs}}$.

Proof is omitted.

Remark 4 *The three transfer theorems of this section can be extended to the case of machines with parameters. For example, let us show that* $\mathrm{PH} = \Sigma^k \Rightarrow \mathrm{PH}_{\mathbb{R}_{vs}} = \Sigma^{k+2}_{\mathbb{R}_{vs}}$. *For any problem* $L \in \mathrm{PH}_{\mathbb{R}_{vs}}$ *there exist parameters* $\alpha_1, \ldots, \alpha_p$ *and a problem* $L' \in \mathrm{PH}^0_{\mathbb{R}_{vs}}$ *such that* $(x_1, \ldots, x_n) \in L$ *iff* $(x_1, \ldots, x_n, \alpha_1, \ldots, \alpha_p) \in L'$. *By Theorem 5,* $L' \in (\Sigma^{k+2}_{\mathbb{R}_{vs}})^0$ *under the assumption* $\mathrm{PH} = \Sigma^k$. *This implies that* $L \in \Sigma^{k+2}_{\mathbb{R}_{vs}}$.

References

1. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer-Verlag, 1998.
2. F. Cucker and P. Koiran. Computing over the reals with addition and order: Higher complexity classes. *Journal of Complexity*, 11:358–376, 1995.
3. H. Fournier and P. Koiran. Are lower bounds easier over the reals ? In *Proc. 30th ACM Symposium on Theory of Computing*, pages 507–513, 1998.
4. H. Fournier and P. Koiran. Lower bounds are not easier over the reals: Inside PH. LIP Research Report 99-21, Ecole Normale Supérieure de Lyon, 1999.
5. P. Koiran. Computing over the reals with addition and order. *Theoretical Computer Science*, 133(1):35–48, 1994.
6. K. Meer. A note on a $P \neq NP$ result for a restricted class of real machines. *Journal of Complexity*, 8:451–453, 1992.
7. F. Meyer auf der Heide. A polynomial linear search algorithm for the n-dimensional knapsack problem. *Journal of the ACM*, 31(3):668–676, 1984.
8. F. Meyer auf der Heide. Fast algorithms for n-dimensional restrictions of hard problems. *Journal of the ACM*, 35(3):740–747, 1988.
9. B. Poizat. *Les Petits Cailloux.* Nur Al-Mantiq Wal-Ma'rifah **3**. Aléas, Lyon, 1995.
10. A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, New-York, 1986.
11. M. Shub and S. Smale. On the intractability of Hilbert's Nullstellensatz and an algebraic version of "P=NP". *Duke Mathematical Journal*, 81(1):47–54, 1996.
12. S. Smale. On the P=NP problem over the complex numbers. Lecture given at the MSRI workshop on Complexity of Continuous and Algebraic Mathematics, November 1998. Lecture on video at www.msri.org.
13. L. G. Valiant. Completeness classes in algebra. In *Proc. 11th ACM Symposium on Theory of Computing*, pages 249–261, 1979.
14. L. G. Valiant. Reducibility by algebraic projections. In *Logic and Algorithmic (an International Symposium held in honour of Ernst Specker)*, pages 365–380. Monographie n° 30 de L'Enseignement Mathématique, 1982.

Unlearning Helps

Ganesh Baliga[1], John Case[2], Wolfgang Merkle[3], and Frank Stephan[3]

[1] Rowan University, Glassboro, NJ 08028, USA, baliga@rowan.edu
[2] University of Delaware, Newark, DE 19716-2586, USA, case@cis.udel.edu
[3] Universität Heidelberg, Mathematisches Institut, Im Neuenheimer Feld 294, 69120 Heidelberg, Germany, merkle|fstephan@math.uni-heidelberg.de

Abstract. Overregularization seen in child language learning, re verb tense constructs, involves abandoning correct behaviors for incorrect ones and later reverting to correct behaviors. Quite a number of other child development phenomena also follow this U-shaped form of learning, unlearning, and relearning. A *decisive* learner doesn't do this and, in general, *never* abandons an hypothesis H for an inequivalent one where it later conjectures an hypothesis equivalent to H. The present paper shows that decisiveness is a real restriction on Gold's model of iteratively (or in the limit) learning of grammars for languages from positive data. This suggests that natural U-shaped learning curves may not be a mere accident in the evolution of human learning, but may be necessary for learning. The result also solves an open problem.
Second-time decisive learners conjecture each of their hypotheses for a language at most twice. By contrast, they are shown not to restrict Gold's model of learning, and correspondingly, there is an apparent lack of reports in child development of the opposite, W-shaped learning curves.

1 Introduction

In developmental and cognitive psychology there are a number of child development phenomena in which the child passes through a sequence of the form: child learns behavior X, child *un*learns X, and then child *re*learns X [13]. This performance is described as U-shaped. For example, we have the important case of $X =$ *regularization in language acquisition* [7]. We explain. In the learning of the proper forms for past tense (say, in English), children learn correct syntactic forms (for example, 'called' with 'call' and 'caught' with 'catch'), then they overregularize and begin to form past tenses by attaching regular verb endings such as 'ed' to the present tense forms (even in irregular cases like 'catch' where that is not correct), and lastly they correctly handle the past tenses (both regular and irregular). We also see U-shaped sequences for child development in such diverse domains as understanding of temperature, understanding of weight conservation, the interaction between understanding of object tracking and object permanence, and face recognition [13]. Within some of these domains we also see temporally separate U-shaped curves for the child's qualitative and quantitative assessments [13].

One wonders if the seemingly inefficient U-shaped sequence of learning, unlearning, and relearning is a mere accident of the natural evolutionary process

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 844–856, 2000.
© Springer-Verlag Berlin Heidelberg 2000

that built us (e.g., language learning) humans or something that must be that way to achieve the learning at all.[1] We do not answer this very difficult empirical question. But, in the present paper, in the context of Gold's formal model of language learning (from positive data) [5], we show there are cases where successful learning *requires* U-shaped sequences of learning, unlearning, and relearning. More precisely, but informally, *decisive learning* [8] is learning in which the learner can*not* (inefficiently) conjecture an hypothesis H_1, then conjecture a behaviorally *in*equivalent hypothesis H_2, and then conjecture an hypothesis H_3 which *is* behaviorally equivalent to H_1. Hence, a decisive learner never returns to abandoned hypotheses and therefore continues to output correct hypotheses from the time it has output its first correct hypothesis. A consequence of our main results in the present paper (Theorem 6 and Corollary 7) is that there are some classes of r.e. languages *learnable* from positive data which can*not* be learned decisively. Hence, there are cases where U-shaped curves, featuring unlearning, are *necessary*. It would be interesting in the future to characterize these cases in a way that provides insight into why we see so many cases of unlearning in child development.

A *text* for a language L is an infinite sequence of all and only the elements of L (together with some possible #'s).[2] A text for L should be thought of as a presentation of the *positive data about* L. Gold's model of language learning from positive data [5] is also called **EX**-*learning from text*. A machine M **EX**-*learns from text* a language L iff (by definition) M, fed any text for L, outputs a sequence of grammars[3], and this sequence eventually converges to some fixed grammar for L. A machine M **BC**-*learns from text* [1,9] a language L iff (by definition) M, fed any text for L, outputs a sequence of grammars, and this sequence eventually converges to nothing but grammars for L.[4]

Our main result, i.e., Theorem 6 in Section 3, shows that there are classes of r.e. languages which can be **EX**-learned from text, but cannot be decisively **BC**-learned from text. From this we obtain in Corollaries 7 and 8 that decisive learning *limits* learning power for **EX**-learning and **BC**-learning from text, respectively. The latter result on **BC**-learning has been nicely shown by Fulk, Jain and Osherson [3], whereas the result on **EX**-learning is apparently new and answers an open question from [8]. Note that it has already been known before that when learning programs for functions, decisiveness does *not* limit learning power, see Remark 10 for references and further explanation.

We informally define *second-time decisive learning* as learning in which, for each text input to the learner, there is *no* conjectured subsequence of hypotheses H_1, H_2, H_3, H_4, H_5 such that H_1 *is* behaviorally equivalent to H_3 and H_5 but

[1] In [7,13] the concern is *not* this evolutionary question. It is, instead, about how humans are actually doing U-shaped learning (and not about why humans bother to employ U-shaped learning in the first place).

[2] The elements of L might occur arbitrarily often and in any order. The #'s represent pauses. The only text for the empty language is an infinite sequence of such pauses.

[3] We have in mind type-0 grammars, or, equivalently, r.e. indices.

[4] **EX**-learning from text involves *syntactic* converge to correct grammars. **BC**-learning from text involves semantic or behaviorally correct convergence.

*in*equivalent to H_2 and H_4.[5] Contrasting interestingly with our main result we show in Proposition 15 in Section 4 that the learning power of second-time decisive **EX**-learners is the same as that of unrestricted **EX**-learners. Hence, the additional power of non-decisive learning is already achieved if we allow the learner to "return" to each abandoned hypothesis at most once. Hence, interestingly coinciding with the apparent lack of reports of W-shaped learning in child development, we see that in Gold's paradigm (generally influential in cognitive science [10]), W-shaped learning is *not* necessary.

We also show (Proposition 16 in Section 4) that **EX**-learnable classes which contain the entire set of natural numbers, $\mathbb{N}$, *do* have a decisive **EX**-learner.[6]

2 Decisive Learning

Next we present the definition of *decisiveness* formally. We use the variable σ (with or without subscripts) for finite initial segments of texts and call them strings. The range of a string σ is the set of non-pauses in σ and is denoted by $\mathrm{rng}(\sigma)$. We write $\preceq$ for the prefix relation between strings and texts, i.e., for example $\sigma_1 \preceq \sigma_2$ just in case σ_1 is a prefix of σ_2. We write $\sigma\tau$ for the concatenation of the strings σ and τ. The index $M(\sigma)$ is machine M's conjectured grammar based on the information contained in σ and $\mathrm{W}_{M(\sigma)}$ is the language defined by the grammar $M(\sigma)$.

Definition 1. *A learner M is* decisive *on a set S of strings iff there are no three strings σ_1, σ_2 and σ_3 such that σ_1 and σ_3 are in S, $\sigma_1 \preceq \sigma_2 \preceq \sigma_3$ and $\mathrm{W}_{M(\sigma_1)}$ differs from $\mathrm{W}_{M(\sigma_2)}$ but is equal to $\mathrm{W}_{M(\sigma_3)}$.*

A learner M is decisive *iff it is decisive on the set of all strings.*

So a decisive learner avoids U-shaped learning curves as discussed in the introduction. We conclude this section with a series of *remarks* and their proofs describing some standard techniques for the construction of decisive learners.

Remark 2. A finite class $\boldsymbol{C}$ can always be decisively **EX**-learned.

In order to obtain a corresponding learner, we fix a total ordering on $\boldsymbol{C}$ which extends the partial ordering given by set theoretic inclusion. For every distinct pair of sets in $\boldsymbol{C}$ we let I contain the least number which is in the larger (in our total ordering) but not in the smaller set. The learner then is equipped with a finite table which contains canonical indices[7] for I and for all sets in $\{C \cap I : C \in \boldsymbol{C}\}$. On input σ, the learner outputs the index of the least set in $\boldsymbol{C}$ which contains the intersection of $\mathrm{rng}(\sigma)$ with I, and, in case there is no such set, retains the preceding index.

[5] Equivalent hypotheses are just grammars generating identical languages; inequivalent hypotheses are grammars generating distinct languages.

[6] A basic trick from [12] is employed, modified, in the proofs of both our Propositions 15 and 16.

[7] A *canonical indexing* [11] numerically encodes for each finite set both a procedure to list it *and* its size (the latter so one knows when the listing is finished).

Remark 3. If a learner M is decisive on two sets S_1 and S_2 of strings such that the classes $\{\mathrm{W}_{M(\sigma)} : \sigma \text{ in } S_1\}$ and $\{\mathrm{W}_{M(\sigma)} : \sigma \text{ in } S_2\}$ are disjoint, then M is actually decisive on the union of S_1 and S_2.

For a proof, assume that there were strings σ_1, σ_2, and σ_3 with $\sigma_1 \preceq \sigma_2 \preceq \sigma_3$ where σ_1 and σ_3 are in the union of S_1 and S_2 and $\mathrm{W}_{M(\sigma_1)}$ is equal to $\mathrm{W}_{M(\sigma_3)}$. In case σ_1 and σ_3 were either both in S_1 or both in S_2, M could not be decisive on the corresponding set S_i, whereas in case one of the strings were in S_1 and the other in S_2, this would contradict the assumption on M, S_1, and S_2.

Remark 4. By delaying the learning process, we can transform a learner M_1 into a new learner M_2 for the same class such that the outputs of M_2 satisfy certain properties. Here on input σ the learner M_2 outputs $M_1(\gamma)$ where γ is the maximal prefix of σ such that M_2 has already been able to verify that $M_1(\gamma)$ has the property under consideration. In the remainder of this remark, we make this idea more precise and we argue that while delaying the learning process this way we can preserve decisiveness.

Formally, we fix a binary computable predicate on strings, written in the form $P_\sigma(\tau)$, such that for all strings σ_1, σ_2, and γ,

$$[P_{\sigma_1}(\gamma) \quad \text{and} \quad \sigma_1 \preceq \sigma_2] \qquad \text{implies} \qquad P_{\sigma_2}(\gamma) \tag{1}$$

and we define a partial function s on strings by

$$s(\sigma) := \max\{\gamma : \gamma \preceq \sigma \text{ and } P_\sigma(\gamma)\}$$

where it is to be understood that $s(\sigma)$ is defined iff the maximization in its definition is over a nonempty set. Then by (1), the function s is nondecreasing in the sense that if s is defined on strings σ_1 and σ_2 with $\sigma_1 \preceq \sigma_2$, then $s(\sigma_1)$ is a prefix of $s(\sigma_2)$.

In case $s(\sigma)$ is defined, we let $M_2(\sigma) = M_1(s(\sigma))$ and, otherwise, we let $M_2(\sigma) = e$ for some fixed index e. We will refer to such a transformation of M_1 by the expression delaying with initial value e and condition P and, informally, we will call the learner M_2 a delayed learner with respect to M_1. For example, in the sequel we will consider delayings with conditions P, firstly, such that $P_\sigma(\tau)$ is true iff the range of τ is contained in $\mathrm{W}_{M_1(\tau),|\sigma|}$ and, secondly, such that $P_\sigma(\tau)$ is true for all σ and τ where the computation of M_1 on input τ terminates in at most $|\sigma|$ steps. The rationale for choosing these condition will become clear in connection with the intended applications.

Now assume that we are given a class $\boldsymbol{C}$ where for every text T for a set in $\boldsymbol{C}$, the values of the function s have unbounded length on the prefixes of T, that is, there are arbitrarily long prefixes τ and σ of T with $\tau \preceq \sigma$ such that $P_\sigma(\tau)$ is true. Then it is immediate from the definition of M_2 that in case the learner M_1 learns $\boldsymbol{C}$ under the criterion **EX** or **BC**, the delayed learner M_2 learns $\boldsymbol{C}$ under the same criterion.

Finally, assume that M_1 is decisive and that either $\mathrm{W}_e \neq \mathrm{W}_{M_1(\sigma)}$ for all strings σ or $\mathrm{W}_e = \mathrm{W}_{M_1(\lambda)}$, where λ denotes the empty string. Exploiting that in both cases by assumption on e the set $E = \{\sigma : \mathrm{W}_{M_2(\sigma)} = \mathrm{W}_e\}$ is closed under taking prefixes, one can show that M_2 is again decisive.

3 The Limits of Decisive Learning

In this section, we show that decisiveness is a proper restriction for **EX**-learning from text. In the proof of the latter result we will use Lemma 5, which relates to a result stated in [6, Exercise 5–3]: every class which does not contain $\mathbb{N}$ and can be **EX**-learned, can in fact be **EX**-learned by a nonexcessive learner. Here a learner M is said to be *nonexcessive* if and only if M never outputs an index for $\mathbb{N}$, that is, for all σ, $\mathrm{W}_{M(\sigma)}$ differs from $\mathbb{N}$. Observe that Lemma 5 can be shown with **BC**-learning replaced by **EX**-learning by essentially the same proof.

Lemma 5. *Let $\boldsymbol{C}$ be an infinite class where every finite set is contained in all but finitely many sets in $\boldsymbol{C}$. If the class $\boldsymbol{C}$ can be decisively* **BC***-learned from text, then it can be decisively* **BC***-learned from text by a nonexcessive learner.*

Proof. By assumption there is a decisive **BC**-learner M_0 which learns $\boldsymbol{C}$ from text. In case M_0 never outputs an index for $\mathbb{N}$ we are done. So fix a string τ_0 such that $\mathrm{W}_{M_0(\tau_0)} = \mathbb{N}$ and, by assumption on $\boldsymbol{C}$, choose $A \neq \mathbb{N}$ in $\boldsymbol{C}$ which contains $\mathrm{rng}(\tau_0)$. For every text for A, the learner M_0 must eventually output an index for A and consequently we can fix an extension τ of τ_0 such that $M_0(\tau)$ is an index for A. But A differs from $\mathbb{N}$ and thus for all extensions of τ, the decisive learner M_0 can never again output an index for $\mathbb{N}$ (whence, in particular, $\mathbb{N} \notin \boldsymbol{C}$). In the construction of a nonexcessive **BC**-learner M as asserted in the lemma, the key idea now is to restrict the output of M to indices of the form $M_0(\tau\sigma)$, except for at most finitely many additional indices of sets in $\boldsymbol{C}$ which do not contain the set $D = \mathrm{rng}(\tau)$. We partition the set of all strings into the sets

$$S_1 = \{\sigma : D \not\subseteq \mathrm{rng}(\sigma)\} \qquad \text{and} \qquad S_2 = \{\sigma : D \subseteq \mathrm{rng}(\sigma)\}$$

and we partition $\boldsymbol{C}$ into the classes

$$\boldsymbol{C}_1 = \{L \text{ in } \boldsymbol{C} \ : D \not\subseteq L\} \qquad \text{and} \qquad \boldsymbol{C}_2 = \{L \text{ in } \boldsymbol{C} \ : D \subseteq L\} \ .$$

By assumption on $\boldsymbol{C}$, the class $\boldsymbol{C}_1$ is finite and we can fix an decisive **EX**-learner M_1 for $\boldsymbol{C}_1$ as in Remark 2, which in particular outputs only indices for sets in $\boldsymbol{C}_1$. Concerning the class $\boldsymbol{C}_2$, first consider a learner $\widetilde{M_2}$ which on input σ outputs $M_0(\tau\sigma)$. We leave to the reader the routine task of showing that $\widetilde{M_2}$ **BC**-learns $\boldsymbol{C}_2$ and inherits the property of being decisive from M. Let M_2 be the learned obtained according to Remark 4 by delaying $\widetilde{M_2}$ with initial index e and condition P where e is an index for some set which neither contains D nor is in $\boldsymbol{C}_1$ and the condition P is defined by

$$P_\sigma(\gamma) \quad \text{iff} \quad D \text{ is contained in } \mathrm{W}_{\widetilde{M_2}(\gamma),|\sigma|} \ .$$

By the discussion in Remark 4, the delayed learner M_2 is again a decisive **BC**-learner for $\boldsymbol{C}_2$. Now we obtain a learner M as required where

$$M(\sigma) \quad = \quad \begin{cases} M_1(\sigma) & \text{in case } \sigma \text{ is in } S_1 \ , \\ M_2(\sigma) & \text{otherwise} \ . \end{cases}$$

In order to see that M **BC**-learns $\boldsymbol{C}$, assume that M is presented to a text T for a set L in $\boldsymbol{C}$. In case L is in $\boldsymbol{C}_1$, the learner M agrees on all prefixes of T with the learner M_1 for $\boldsymbol{C}_1$ while, similarly, in case L is in $\boldsymbol{C}_2$, the learner M agrees on almost all prefixes of T with the learner M_2 for $\boldsymbol{C}_2$. In order to see that M is decisive it suffices to observe that S_1 and S_2 witness that M satisfies the assumption of Remark 3. The latter holds because M_1 and M_2 are both decisive and because every set of the form $\mathrm{W}_{M_2(\sigma)}$ contains D, whereas no set of the form $\mathrm{W}_{M_1(\sigma)}$ contains D. □

Theorem 6 and Corollary 7 are the main results of this paper.

Theorem 6. *There is a class which can be* **EX**-*learned from text, but cannot be decisively* **BC**-*learned from text.*

From Theorem 6 the following corollaries are immediate. Here Corollary 7 answers an open problem stated in [8] and [3], while Corollary 8 has been previously shown by Fulk, Jain and Osherson [3]. We will argue in Remark 17 that the proof of Corollary 8 in [3] neither yields Theorem 6 nor Corollary 7.

Corollary 7. *The concept of decisive* **EX**-*learning from text is a proper restriction of* **EX**-*learning from text, that is, there is a class which can be* **EX**-*learned from text, but cannot be decisively* **EX**-*learned from text.*

Corollary 8. *The concept of decisive* **BC**-*learning from text is a proper restriction of* **BC**-*learning from text, that is, there is a class which can be* **BC**-*learned from text, but cannot be decisively* **BC**-*learned from text.*

Proof of the theorem. In order to construct a class $\boldsymbol{C}$ as required we define, for the scope of this proof, for all subsets A of $\mathbb{N}$ a notion of id by

$$\mathrm{id}(A) := \min\{m \text{ in } \mathbb{N} \cup \{\infty\} : m \text{ is not in } A\} \ .$$

In terms of the id of a set we can already now summarize the features of the class $\boldsymbol{C}$ which are relevant for making it **EX**-learnable. We will have

$$\boldsymbol{C} \ = \ \boldsymbol{H} \cup \{\mathrm{W}_{g(m)} : m \text{ in } \mathbb{N}\} \qquad \text{with} \qquad \boldsymbol{H} \ = \ \{\mathrm{rng}(\sigma) : \sigma \text{ in } Z\} \tag{2}$$

where Z and g are defined below such that they have the following properties. The set Z is recursively enumerable and the function g is computable in the limit, that is, there is a computable function $\widetilde{g}$ in two arguments such that $g(m)$ is equal to $\lim_{s\to\infty} \widetilde{g}(m,s)$. Furthermore for all m, the set $\mathrm{W}_{g(m)}$ has id m and there are at most finitely many σ in Z where $\mathrm{rng}(\sigma)$ has id m.

Claim 1. The class $\boldsymbol{C}$ can be **EX**-learned from text.

Proof. We construct a learner M which **EX**-learns $\boldsymbol{C}$ from text. Given a language L of id m in $\boldsymbol{C}$, let $\boldsymbol{H}_m$ contain the sets of id m in $\boldsymbol{H}$. By construction of $\boldsymbol{C}$, the number m and the class $\boldsymbol{H}_m$ are both finite and L must be equal to $\mathrm{W}_{g(m)}$ or to one of the sets in $\boldsymbol{H}_m$. Moreover, for every text for L almost all prefixes of the text have id m and, by assumption on Z and g, from m we can compute

in the limit $g(m)$ and a list of canonical indices for the members of $\boldsymbol{H}_m$. Thus we can simply assume that M has access to m, $g(m)$, and $\boldsymbol{H}_m$, as long as we ensure that M is defined even on prefixes of the given text for L for which its approximations to these values are not yet correct. So the learner M can simply output some fixed index for $\mathrm{rng}(\sigma)$ in case this set is in $\boldsymbol{H}_m$ while, otherwise, M outputs $g(m)$. □

Claim 2. If $\boldsymbol{C}$ can be decisively **BC**-learned, then it can be decisively **BC**-learned by a primitive recursive learner.

Proof. Assume that a learner N_1 **BC**-learns $\boldsymbol{C}$ decisively. Consider the delayed learner N_2 obtained by delaying N_1 with initial index $N_1(\lambda)$ and condition P where $P_\sigma(\tau)$ is true iff the computation of N_1 on input τ terminates after at most $|\sigma|$ steps. Then N_2 is again a **BC**-learner for $\boldsymbol{C}$ which, by Remark 4, is again decisive. Moreover, N_2 can obviously be chosen to be primitive recursive. □

Fix an enumeration $M_0, M_1, \ldots$ of all primitive recursive learners such that $M_i(\sigma)$ is uniformly computable in i and σ. According to Claim 2, in order to achieve that $\boldsymbol{C}$ cannot be decisively **BC**-learned, it suffices to ensure by diagonalization that for all i, the class $\boldsymbol{C}$ is not decisively **BC**-learned by M_i. While diagonalizing against M_i we will exploit that for every string σ with

$$\mathrm{rng}(\sigma) \subsetneqq \mathrm{W}_{M_i(\sigma)} \ , \tag{3}$$

if the learner M_i learns $\mathrm{rng}(\sigma)$ and $\mathrm{W}_{M_i(\sigma)}$, then it cannot be decisive. In order to see the latter observe that by extending σ to a text for $\mathrm{rng}(\sigma)$ we eventually reach a string τ where $M_i(\tau)$ is an index for $\mathrm{rng}(\sigma)$, while on every text for $\mathrm{W}_{M_i(\sigma)}$ which extends τ, the learner M_i converges to nothing but indices for the previously abandoned guess $\mathrm{W}_{M_i(\sigma)}$.

For the scope of this proof, a string σ which satisfies (3) will be called an i-witness. During the construction we try to diagonalize, for all indices i, against the learner M_i by putting $\mathrm{rng}(\sigma)$ and $\mathrm{W}_{M_i(\sigma)}$ into $\boldsymbol{C}$ for some i-witness σ. Here, however, we have to observe that the remaining choices in the definition of the class $\boldsymbol{C}$ amount to specify a set Z and a function g which have the properties stated above.

We fix an effective enumeration of all pairs (σ, i) such that σ is an i-witness. For every i, we let Z_i be the (possibly finite or even empty) set such that for every natural number $l \geq i$, among all the i-witnesses with range of id l, the set Z_i contains the one which is enumerated first and we let Z be the union of the sets Z_i. Then Z is recursively enumerable by construction. Moreover, for every m, there are at most finitely many strings σ in Z such that $\mathrm{rng}(\sigma)$ has id m because each Z_i contains at most one such witness and every set Z_i with $i > m$ contains no such witness. From the definition of the concept i-witness it is immediate that for all i,

$$\mathrm{id}(\mathrm{rng}(\sigma)) \leq \mathrm{id}(\mathrm{W}_{M_i(\sigma)}) \quad \text{for all } \sigma \text{ in } Z_i \ . \tag{4}$$

We have arranged by the definition of Z that the class $\boldsymbol{C}$ contains $\mathrm{rng}(\sigma)$ for all i and all witnesses in Z_i. Thus in order to diagonalize against the learner M_i it

suffices to put $\mathrm{W}_{M_i(\sigma)}$ into $\boldsymbol{C}$ for some σ in Z_i, where by definition of $\boldsymbol{C}$ this can be done by ensuring that the index $M_i(\sigma)$ appears in the range of g. Here, however, for every m, we can only put a single set $\mathrm{W}_{g(m)}$ of id m into $\boldsymbol{C}$. In order to define g accordingly, we let for all i,

$$E_i := \{\mathrm{id}(\mathrm{W}_{M_i(\sigma)}) : \sigma \text{ in } Z_i\}$$

be the set of all ids which are realized by sets of the form $\mathrm{W}_{M_i(\sigma)}$ with σ in Z_i. Moreover, we fix a recursive function $k : \mathbb{N}^3 \to \mathbb{N}$ such that for all i and m where m is in E_i, firstly, the limit

$$\kappa(i,m) \;=\; \lim_{s\to\infty} k(i,m,s)$$

exists, secondly, $\kappa(i,m) = M_i(\sigma)$ for some σ in Z_i and, thirdly, $\mathrm{W}_{\kappa(i,m)}$ has id m. Such a function k can be constructed by running on input i, m, and s for s steps some fixed procedure which enumerates in parallel the sets $\mathrm{W}_{M_i(\sigma)}$ with σ in Z_i and outputs the index $M_i(\sigma)$ which first enumerated all numbers less than m but hasn't yet enumerated m itself. Next we define sets S_m and a function h by a straightforward priority construction, which, however, is non-effective

$$\begin{aligned} S_m &= \{i \,:\, m \text{ in } E_i \text{ and } h(l) \neq i \text{ for all } l < m\}, \\ h(m) &= \begin{cases} \min S_m & \text{if } S_m \neq \emptyset, \\ * & \text{otherwise.} \end{cases} \end{aligned}$$

Intuitively speaking, the set S_m contains the indices i such that, firstly, we have not yet diagonalized explicitly against learner M_i and, secondly, we have a chance to diagonalize now because Z_i contains a witness σ such that $\mathrm{W}_{M_i(\sigma)}$ has id m. Moreover, for all i in S_m, such an index $M_i(\sigma)$ is given by $\kappa(i,m)$. We pick an appropriate easily computable index e_m for the set $\mathbb{N} \setminus \{m\}$ and define

$$g(m) \;:=\; \begin{cases} \kappa(h(m),m) & \text{if } h(m) \neq *, \\ e_m & \text{otherwise.} \end{cases}$$

Claim 3. The function g is computable in the limit.

Proof. By definition, $g(m)$ is computable in the limit from $h(m)$ and m, whence it suffices to show that the function h is computable in the limit. Now for given m, the inductive definition of h on $0, \ldots, m$ depends only on the membership of numbers less than or equal to m in the sets $E_0, E_1, \ldots$. Furthermore, each set E_i contains exactly the ids of sets of the form $\mathrm{W}_{M_i(\sigma)}$ with σ in Z_i. Now for every string σ in Z_i, the id of $\mathrm{rng}(\sigma)$ and hence by (4) also the id of $\mathrm{W}_{M_i(\sigma)}$ is bounded from below by i, whence the set E_i does not contain numbers less than i. In summary, we can compute $h(m)$ if we are given canonical indices of the sets $F_0, \ldots, F_m$ with $F_i = E_i \cap \{0, \ldots, m\}$. Here, by the definition of E_i, the set F_i contains exactly the numbers $l \leq m$ such that for some σ in Z_i the id of $\mathrm{W}_{M_i(\sigma)}$ is l and by (4) the range of such a string σ has id at most m. Now

by definition of the sets Z_i, we can effectively in i enumerate the finitely many σ in Z_i such that $\mathrm{rng}(\sigma)$ has id at most m. Moreover, for each such σ, the ids of the sets $\mathrm{W}_{M_i(\sigma),0}, \mathrm{W}_{M_i(\sigma),1}, \ldots$ converge to the, possibly infinite, id of $\mathrm{W}_{M_i(\sigma)}$, whence in the limit this approximation converges to the finite value $\mathrm{id}(M_i(\sigma))$ or reveals that the latter value is strictly larger than m. So we are done because by combining the two approximation processes just described we obtain an effective procedure which on input i converges to a canonical index for F_i. □

Claim 4. Let i be such that E_i is infinite. Then the class $\boldsymbol{C}$ contains a set of the form $\mathrm{W}_{M_i(\sigma)}$ with σ in Z_i, whence in particular M_i cannot learn $\boldsymbol{C}$ decisively from text.

Proof. If there is an i-witness σ in Z_i such that $\boldsymbol{C}$ contains $\mathrm{W}_{M_i(\sigma)}$, then by construction the class $\boldsymbol{C}$ contains also $\mathrm{rng}(\sigma)$ and hence cannot be learned decisively by M_i. So if there is an m with $h(m) = i$ then $g(m)$ is equal to $M_i(\sigma)$ for some σ in Z_i and we are done by definition of $\boldsymbol{C}$. But assuming that $h(m)$ differs from i for all m yields a contradiction because in this case $h(m)$ is by definition an element of $\{0, \ldots, i-1\}$ for all m in the infinite set E_i, while on the other hand the function h attains every value in $\mathbb{N}$ at most once. □

Claim 5. The set $\{\mathrm{id}(L) : L \in \boldsymbol{C} \text{ and } L \text{ infinite}\}$ is infinite.

Proof. It suffices to show that for every given l, there is an infinite set in $\boldsymbol{C}$ which has finite id larger than l. We fix an index i such that on input σ, the learner M_i outputs an index for the set $\mathbb{N} \setminus \{m_\sigma\}$ where m_σ is the least number which is strictly greater than l and all the numbers in $\mathrm{rng}(\sigma)$. Then every string σ is an i-witness where the set $\mathrm{W}_{M_i(\sigma)}$ has finite id m_σ. Thus the set E_i is infinite, whence by Claim 4 the class $\boldsymbol{C}$ contains a set of the form $\mathrm{W}_{M_i(\sigma)}$ as required. □

Claim 6. Let i be such that M_i **BC**-learns the class $\boldsymbol{C}$ from text. Then the set Z_i is infinite.

Proof. Fix an arbitrary natural number m_0 and by Claim 5, let $L \neq \mathbb{N}$ be an infinite set in $\boldsymbol{C}$ of id $m \geq m_0$. On every given text T for L, the learner M_i eventually converges to nothing but indices for L, whence almost all prefixes of T are i-witnesses with range of id m. Now m_0 has been chosen arbitrarily and consequently there are i-witnesses with range of arbitrarily large id, whence the set Z_i is infinite by its definition. □

Claim 7. Let i be such that M_i does never output an index for $\mathbb{N}$. If Z_i is infinite, then also E_i is infinite.

Proof. By construction, the strings in Z_i all have different id, whence in case Z_i is infinite, the set $\{\mathrm{id}(\mathrm{rng}(\sigma)) : \sigma \in Z_i\}$ is infinite, too. Then the set E_i, which by definition contains just the ids of the sets $\mathrm{W}_{M_i(\sigma)}$ with σ in Z_i, must contain arbitrarily large values because of (4). Hence the set E_i is infinite because by assumption on M_i the sets of the form $\mathrm{W}_{M_i(\sigma)}$ all have finite id. □

Now assume for a contradiction that the class $\boldsymbol{C}$ can be decisively **BC**-learned from text. Then by Lemma 5 and the proof of Claim 2, this fact would be witnessed by an decisive learner M_i which never outputs an index for $\mathbb{N}$. Thus the sets Z_i and E_i are both infinite by Claims 6 and 7, whence by Claim 4 the learner M_i cannot learn $\boldsymbol{C}$ as assumed. □

Theorem 6 above and the following Remark 9 show that the concepts of **EX**-learning from text and decisive **BC**-learning from text are incomparable in the sense that for each of these concepts there are classes which can be learned under this concept but not under the other one. In Remark 9, which we state without proof, we exploit that one of the standard constructions of a class which can be **BC**- but not **EX**-learned from text actually yields a class which can be decisively **BC**-learned from text.

Remark 9. The class of all sets of the form $K \cup D$ where K is the halting problem and D is a finite subset of $\mathbb{N}$ can be decisively **BC**-learned from text, but cannot be **EX**-learned from text.

Remark 10. Schäfer-Richter (see [12] and [8, Section 4.5.5]) showed that every class of functions which can be **EX**-learned can in fact also be decisively **EX**-learned. The same holds for **BC**-learning as shown by Fulk, Jain, and Osherson [3] and, implicitly, by Freivalds, Kinber, and Wiehagen [4].

4 The Power of Decisive Learning

While we have shown in the last section that decisiveness properly restricts **EX**-learning, we will show now that in certain respects decisive and general **EX**-learning are rather close. We show that every **EX**-learnable class **C** can be learned under a criterion which is slightly more liberal than decisive **EX**-learning in so far as every abandoned hypothesis can be "reconjectured" at most once and that, furthermore, **C** can indeed be decisively **EX**-learned if it contains $\mathbb{N}$.

Definition 11. *A learner M is* second-time decisive *iff there are no five strings $\sigma_1, \ldots, \sigma_5$ with $\sigma_1 \preceq \sigma_2 \preceq \sigma_3 \preceq \sigma_4 \preceq \sigma_5$ such that $\mathrm{W}_{M(\sigma_1)}$ is equal to $\mathrm{W}_{M(\sigma_3)}$ and $\mathrm{W}_{M(\sigma_5)}$ but differs from $\mathrm{W}_{M(\sigma_2)}$ and $\mathrm{W}_{M(\sigma_4)}$.*

So a second-time decisive learner avoids W-shaped learning but in general may show U-shaped learning. Due to lack of space, we omit the proof of Remark 12.

Remark 12. A learner M is second-time decisive if and only if there is a set S of strings such that M is decisive on S, as well as on the complement of S.

In connection with Lemma 14 below we recall the concept of locking sequence.

Definition 13. *Let a learner M, a set L and a string σ be given. Then σ is a* locking sequence for M and L *iff* $\mathrm{rng}(\sigma)$ *is contained in $L = \mathrm{W}_{M(\sigma)}$ and $M(\sigma\tau)$ is equal to $M(\sigma)$ for all strings τ over $\mathrm{W}_{M(\sigma)}$. Furthermore, σ is a* locking sequence for M *iff σ is a locking sequence for M and $\mathrm{W}_{M(\sigma)}$.*

A learner M learns via locking sequences *iff every text for a language L which is* **EX**-*learned by M has a prefix which is a locking sequence for M and L.*

The subsequent proofs in this section will use Lemma 14 below. Our proof of this lemma is an adaptation of the proof of the well-known fact that every class which can be **EX**-learned from text at all, can actually be **EX**-learned from text via locking sequences (see for example Theorem 13 in [2] and the references cited there). To save space, we omit the proof of Lemma 14.

Lemma 14. *Let g be a computable function such that every finite set is contained in infinitely many of the pairwise distinct sets $\mathrm{W}_{g(0)}, \mathrm{W}_{g(1)}, \ldots$ and let the class $\boldsymbol{C}$ be* **EX***-learned from text by some learner M_0.*

Then there is a learner M and a set S such that M **EX***-learns $\boldsymbol{C}$ from text via locking sequences, M is decisive on S and on its complement and*

$$\mathrm{W}_{M(\sigma)} \text{ is in } \begin{cases} \{\mathrm{W}_{g(i)} : i \text{ in } \mathbb{N}\} & \text{if } \sigma \text{ is in } S\ , \\ \{\mathrm{W}_{M_0(\tau)} : \tau \text{ is a string}\} & \text{if } \sigma \text{ is not in } S\ . \end{cases} \tag{5}$$

We have seen in Sect. 3 that there are classes which can be **EX**-learned from text, but cannot be learned so decisively. Proposition 15 shows that the additional power of non-decisive learning is already achieved if we allow the learner to "return" to each abandoned hypothesis at most once.

Proposition 15. *Every class which can be* **EX***-learned from text can also be* **EX***-learned from text by a second-time decisive learner.*

Proof. By Remark 12, Proposition 15 is a special case of Lemma 14 where we fix a computable function g such that for all i, the set $\mathrm{W}_{g(i)}$ is just $\{0, \ldots, i\}$. □

The class constructed in the proof of Theorem 6 in order to separate the concepts of general and decisive **EX**-learning does not contain the set $\mathbb{N}$. By Proposition 16, which is again a direct consequence of Lemma 14, this is no coincidence.

Proposition 16. *Every class which contains $\mathbb{N}$ and can be* **EX***-learned from text can be decisively* **EX***-learned from text.*

Proof. Let $\boldsymbol{C}$ be **EX**-learnable and contain $\mathbb{N}$. Fulk [2] showed that $\boldsymbol{C}$ has a prudent learner M_0 – such a learner outputs only indices of sets which it also learns. Since there is a locking-sequence τ for $\mathbb{N}$, M_0 does not identify any finite language containing $\mathrm{rng}(\tau)$. Thus, defining $W_{g(i)} = \{0, 1, \ldots, i + \max \mathrm{rng}(\tau)\}$ makes the sets $W_{g(i)}$ different from all sets learned by M_0 and thus also different from all sets conjectured by M_0.

We apply Lemma 14 to M_0 and g. We obtain an **EX**-learner M for $\boldsymbol{C}$ and a set S of strings such that M is decisive on S and its complement. Moreover, $W_{M(\sigma)}$ and $W_{M(\eta)}$ are different for all $\sigma \in S$ and $\eta \notin S$, whence M is already decisive by Remark 3. □

Remark 17. Fulk, Jain and Osherson [3, Theorem 4] give an example of a class $\boldsymbol{L}$ which can be **BC**-learned from text but cannot be learned so decisively. While their construction bears some similarities to the construction of the class

$\boldsymbol{C}$ in the proof of Theorem 6, their class $\boldsymbol{L}$ is not **EX**-learnable from text and consequently neither yields Theorem 6 nor a separation of decisive and general **EX**-learning as stated in Corollary 7.

For a proof, note that there is no set in $\boldsymbol{L}$ which contains the numbers $\langle 0,0\rangle$ and $\langle 1,0\rangle$, whence by switching to some fixed index for $\mathbb{N}$ as soon as the data contains both numbers, every **EX**-learner for $\boldsymbol{L}$ can be transformed into an **EX**-learner for $\boldsymbol{L}$ which also identifies $\mathbb{N}$. But then by Proposition 16, if the class $\boldsymbol{L}$ were **EX**-learnable, it were decisively **EX**-learnable, whereas by construction $\boldsymbol{L}$ is not even decisively **BC**-learnable.

Acknowledgements

We would like to thank Rolf Wiehagen for helpful comments and for pointing out the relations to [4] as stated in Remark 10. Moreover, we are grateful to an anonymous referee of the International Colloquium on Automata, Languages and Programming 2000 for a most helpful list of corrections.

References

1. John Case and Chris Lynes. Machine inductive inference and language identification. In M. Nielsen and E. M. Schmidt, editors, *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science 140, pages 107–115. Springer-Verlag, 1982.
2. Mark A. Fulk. Prudence and other conditions on formal language learning. *Information and Computation*, 85:1–11, 1990.
3. Mark A. Fulk, Sanjay Jain, and Daniel N. Osherson. Open problems in "Systems that learn". *Journal of Computer and System Sciences*, 49:589–604, 1994.
4. Rusins Freivalds, Efim B. Kinber, and Rolf Wiehagen. On the Power of Inductive Inference from Good Examples. *Theoretical Computer Science*, 110:131–144, 1993.
5. E. Mark Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
6. Sanjay Jain, Daniel N. Osherson, James S. Royer, and Arun Sharma. *Systems that Learn: An Introduction to Learning Theory.* MIT Press, Cambridge, Massachusetts, 1999. Second Edition of Reference [8].
7. Gary Marcus, Steven Pinker, Michael Ullman, Michelle Hollander, T. John Rosen, and Fei Xu. *Overregularization in Language Acquisition.* Monographs of the Society for Research in Child Development, vol. 57, no. 4. University of Chicago Press, 1992. Includes commentary by Harold Clahsen.
8. Daniel N. Osherson, Michael Stob, and Scott Weinstein. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists.* MIT Press, Cambridge, Massachusetts, 1986.

9. Daniel Osherson and Scott Weinstein. Criteria of language learning. *Information and Control*, 52:123–138, 1982.
10. Steven Pinker. Formal models of language learning. *Cognition*, 7:217–283, 1979.
11. Hartley Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw-Hill, New York, 1967. Reprinted, MIT Press 1987.
12. Gisela Schäfer-Richter. Über Eingabeabhängigkeit und Komplexität von Inferenzstrategien. Ph.D. Thesis, Rheinisch-Westfälische Technische Hochschule Aachen, Germany, 1984.
13. Sidney Strauss and Ruth Stavy, editors. *U-Shaped Behavioral Growth.* Developmental Psychology Series. Academic Press, NY, 1982.

Fast Approximation Schemes for Euclidean Multi-connectivity Problems *

(Extended Abstract)

Artur Czumaj[1**] and Andrzej Lingas[2]

[1] Department of Computer and Information Science, New Jersey Institute of Technology, Newark, NJ 07102-1982, USA. czumaj@cis.njit.edu

[2] Department of Computer Science, Lund University, Box 118, S-22100 Lund, Sweden. Andrzej.Lingas@cs.lth.se

Abstract. We present new polynomial-time approximation schemes (PTAS) for several basic minimum-cost multi-connectivity problems in geometrical graphs. We focus on low connectivity requirements. Each of our schemes either significantly improves the previously known upper time-bound or is the first PTAS for the considered problem.

We provide a randomized approximation scheme for finding a biconnected graph spanning a set of points in a multi-dimensional Euclidean space and having the expected total cost within $(1+\varepsilon)$ of the optimum. For any constant dimension and ε, our scheme runs in time $\mathcal{O}(n \log n)$. It can be turned into Las Vegas one without affecting its asymptotic time complexity, and also efficiently derandomized. The only previously known truly polynomial-time approximation (randomized) scheme for this problem runs in expected time $n \cdot (\log n)^{\mathcal{O}((\log\log n)^9)}$ in the simplest planar case. The efficiency of our scheme relies on transformations of nearly optimal low cost special spanners into sub-multigraphs having good decomposition and approximation properties and a simple subgraph connectivity characterization. By using merely the spanner transformations, we obtain a very fast polynomial-time approximation scheme for finding a minimum-cost k-edge connected multigraph spanning a set of points in a multi-dimensional Euclidean space. For any constant dimension, ε, and k, this PTAS runs in time $\mathcal{O}(n \log n)$.

Furthermore, by showing a low-cost transformation of a k-edge connected graph maintaining the k-edge connectivity and developing novel decomposition properties, we derive a PTAS for Euclidean minimum-cost k-edge connectivity. It is substantially faster than that previously known.

Finally, by extending our techniques, we obtain the first PTAS for the problem of Euclidean minimum-cost Steiner biconnectivity. This scheme runs in time $\mathcal{O}(n \log n)$ for any constant dimension and ε. As a byproduct, we get the first known non-trivial upper bound on the number of Steiner points in an optimal solution to an instance of Euclidean minimum-cost Steiner biconnectivity.

* Research supported in part by the DFG Sonderforschungsbereich 376, by DFG grant Me872/7-1, and by TFR grant 96-278. Full version of the paper is available from the authors.

** Work partly done while the author was with the Heinz Nixdorf Institute and Dept. of Mathematics & Computer Science at the University of Paderborn, D-33095 Paderborn, Germany.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 856–868, 2000.
© Springer-Verlag Berlin Heidelberg 2000

1 Introduction

Multi-connectivity graph problems are central in algorithmic graph theory and have numerous applications in computer science and operation research [2,10,22]. They are also very important in the design of networks that arise in practical situations [2,10]. Typical application areas include telecommunication, computer and road networks. Low degree connectivity problems for geometrical graphs in the plane can often closely *approximate* such practical connectivity problems (see, e.g., the discussion in [10,22]).

In this paper, we provide a thorough *theoretical study* of these problems in Euclidean space (i.e., for geometrical graphs). We consider several basic connectivity problems of the following form: for a given set S of n points in the Euclidean space $\mathbb{R}^d$, find a minimum-cost subgraph of a complete graph on S that satisfies a priori given connectivity requirements. The cost of such a subgraph is equal to the sum of the Euclidean distances between adjacent vertices.

The most classical problem we investigate is the *(Euclidean) minimum-cost k-vertex connected spanning subgraph problem*. We are given a set S of n points in the Euclidean space $\mathbb{R}^d$ and the aim is to find a minimum-cost k-vertex connected graph spanning points in S (i.e., a subgraph of the complete graph on S). By substituting the requirement of k-edge connectivity for that of k-vertex connectivity, we obtain the corresponding *(Euclidean) minimum-cost k-edge connected spanning subgraph problem*. We term the generalization of the latter problem which allows for parallel edges in the output graph spanning S as the *(Euclidean) minimum-cost k-edge connected spanning sub-multigraph problem*.

The concept of minimum-cost k-connectivity naturally extends to include that of *Euclidean Steiner k-connectivity* by allowing the use of additional vertices, called *Steiner points*. The problem of *(Euclidean) minimum-cost Steiner k-vertex- (or, k-edge-) connectivity* is to find a minimum-cost graph on a *superset* of the input point set S in $\mathbb{R}^d$ which is k-vertex- (or, k-edge-) connected with respect to S. For $k = 1$, it is simply the famous *Steiner minimal tree* (SMT) problem, which has been very extensively studied in the literature (see, e.g., [11,16]).

Since all the aforementioned problems are known to be $\mathcal{NP}$-hard when restricted to even two-dimensions for $k \geq 2$ [8,18], we focus on efficient constructions of good approximations. We aim at developing a *polynomial-time approximation scheme*, a *PTAS*. This is a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for each fixed $\varepsilon > 0$, $\mathcal{A}_\varepsilon$ runs in time polynomial in the size of the input and produces a $(1+\varepsilon)$-approximation [15].

Previous work. Despite the practical relevance of the multi-connectivity problems for geometrical graphs and the vast amount of practical heuristic results reported (see, e.g., [9,10,22,23]) very little theoretical research has been done towards developing efficient approximation algorithms for these problems. This contrasts with the very rich and successful theoretical investigations of the corresponding problems in general metric spaces and for general weighted graphs (see, e.g., [10,12,15,17]). Even for the simplest (and most fundamental) problem considered in our paper, that of finding a minimum-cost biconnected graph spanning a given set of points in the Euclidean plane, for a long time obtaining approximations achieving better than a $\frac{3}{2}$ ratio had been elusive and only very recently has a PTAS been developed [6]. For any fixed $\varepsilon > 0$, this algorithm outputs

a $(1+\varepsilon)$-approximation in expected time $n\,(\log n)^{\mathcal{O}((\log\log n)^9)}$. The approximation scheme developed in [6] can be extended to arbitrary k and d, but in the general case the dependence on k and particularly on d makes the algorithm impractical. For an $\varepsilon > 0$, the algorithm runs in expected time $n \cdot 2^{(\log\log n)^{2\,d+\binom{2\,d}{2}}\cdot((\mathcal{O}(d\,k^2/\varepsilon))^d\,\log(\varepsilon^{-1}))!}$. Note that d plays a large role in the running time of these schemes. In fact, the result from [6] implies that for every $d = \Omega(\log n)$, even for $k = 2$ no PTAS exists unless $\mathcal{P} = \mathcal{NP}$. Thus, the problem of finding a minimum-cost biconnected spanning subgraph does not have a PTAS (unless $\mathcal{P} = \mathcal{NP}$) even in the metric case. Hence, our restriction to Euclidean graphs in low dimensions plays an essential role in these schemes.

A related, but significantly weaker result has been also presented in [5]. Here an optimal solution to the problem without allowing Steiner points is approximated to an arbitrarily close degree via the inclusion of Steiner points.

When Steiner points are allowed in the minimum-cost Steiner k-vertex- (or k-edge-) connectivity problem, the only non-trivial results are known for $k = 1$, i.e., for the minimum Steiner tree problem (SMT). In the breakthrough paper [3], Arora designed a PTAS for SMT for all constants d. Mitchell independently obtained a similar result for $d = 2$ [19]. Soon after Rao and Smith [20] offered a significantly faster PTAS for SMT running in time $\mathcal{O}(n\,\log n)$ for a constant d. For $k \geq 2$, the only result we are aware of is a $\sqrt{3}$-approximation in polynomial-time for $k = 2$ [14].

New results. In this paper we present new polynomial-time approximation schemes for several of the aforementioned connectivity problems in geometric graphs. We focus on low connectivity requirements. Each of our approximation schemes either significantly improves the previously known upper time-bound or is the first PTAS for the considered problem.

Our *main new result* is a fast polynomial-time (randomized) approximation scheme for finding a biconnected graph spanning a set of points in a d-dimensional Euclidean space and having expected cost within $(1+\varepsilon)$ of optimum. For any constant d and ε, our algorithm runs in expected time $\mathcal{O}(n\,\log n)$. Our scheme is a PTAS for the problem in $\mathbb{R}^d$ for all d such that $2^{d^c\,d^2} = \mathrm{poly}(n)$, for some absolute constant c. We can turn our randomized scheme into a Las Vegas one without affecting its asymptotic time complexity. With a very slight increase of the running time (a constant factor provided that d and ε are constant) we can also obtain a deterministic $(1+\varepsilon)$-approximation. Our scheme is significantly, i.e., by a factor of at least $(\log n)^{\mathcal{O}((\log\log n)^9)}$, faster than that from [6].

Since a minimum-cost biconnected graph spanning a set of points in a metric space is also a minimum-cost two-edge connected graph spanning this set, our PTAS yields also the corresponding PTAS for the Euclidean minimum-cost two-edge connectivity.

We extend the techniques developed for the biconnectivity algorithms and present a fast randomized PTAS for finding a minimum cost k-edge connected *multigraph* spanning a set of points in a d-dimensional Euclidean space. The running time of our Las Vegas scheme is $\mathcal{O}(n\,\log n) + n\,2^{k^{\mathcal{O}(k)}}$ for any constant d and ε.

We are also able to improve upon the k-edge connectivity results from [6] significantly. By showing a low-cost transformation of a k-edge connected graph maintaining the k-edge connectivity and developing novel decomposition properties, we derive a

PTAS for Euclidean minimum-cost k-edge connectivity which for any constant d, ε, and k, runs in expected time $n\,(\log n)^{O(1)}$. The corresponding scheme in [6] requires $n\,(\log n)^{O((\log\log n)^9)}$ time when $d = 2$ and for any constant ε, k.

Furthermore, we present a series of new structural results about minimum-cost biconnected Euclidean Steiner graphs, e.g., a decomposition of a minimum-cost biconnected Steiner graph into minimal Steiner trees. We use these results to derive the *first* PTAS for the minimum-cost Steiner biconnectivity and Steiner two-edge connectivity problems. For any constant d and ε, our scheme runs in expected time $\mathcal{O}(n \log n)$. As a byproduct of the aforementioned decomposition, we also obtain the first known non-trivial upper bound on the minimum number of Steiner points in an optimal solution to an n-point instance of Euclidean minimum-cost Steiner biconnectivity, which is $3n - 2$.

Techniques. The only two known PTAS approaches to Euclidean minimum-cost k-vertex- (or, k-edge-) connectivity (see [5,6]) are based on decompositions of k-connected Euclidean graphs combined with the general framework proposed recently by Arora [3] for designing PTAS for Euclidean versions of TSP, Minimum Steiner Tree, Min-Cost Perfect Matching, k-TSP, etc. (For another related framework for geometric PTAS see [19].) In contrast to all previous applications of Arora's framework using Steiner points in the so-called patching procedures [3,5], a patching method free of Steiner points is given [6]. (Steiner points of degree at least three are difficult to remove for k-connectivity when $k \geq 2$. This should be compared to the problems considered by Arora [3] and Rao and Smith [20] where the output graphs have very simple connectivity structure.) This disallowance in [6] makes it hard to prove strong global structural properties of close approximations with respect to a given geometric partition.

Structural theorems in Arora's framework typically assert the existence of a recursive partition of a box containing the n input points (perturbed to nearest grid points) into cubes such that the optimal solution can be closely approximated by a so called (m, r)-light solution in which, for every cube, there are very few edges crossing its boundaries. The structural theorem in [6] yields only weaker structural properties of approximate solutions ((m, r)-grayness and (m, r)-blueness) which bound solely the number of crossings between the cube boundaries and the edges having exactly one endpoint within the cube. That bound is constant for "short edges" (i.e., edges having length within a constant factor of the side-length of the cube) and it is $\mathcal{O}(\log\log n)$ for "long edges" (assuming that k, d, and ε are constant). Furthermore, most of the crossings are located in one of $2\,d$ prespecified points. The weaker structural properties (especially the fact that there might be as many as $\Theta(\log\log n)$ edges having exactly one endpoint in a cube in the partition) lead to the high time complexity of the main dynamic programming procedure in the PTAS presented in [6].

We take a novel approach in order to guarantee stronger structural properties of approximate solutions disallowing Steiner points. Our approach is partly inspired by the recent use of spanners to speed-up PTAS for Euclidean versions of TSP by Rao and Smith [20]. In effect, for $k = 2$, we are able to prove a substantially stronger structural property (r-local-lightness, see Theorem 3.1) than that in [6]. It yields a constant upper bound on the number of long edges with exactly one endpoint in a cube in the partition provided that d and ε are constant.

Our proof relies on a series of transformations of a $(1+\delta)$-spanner for the input point set, having low cost and the so called *isolation property* [1], into an r-locally-light k-edge connected multigraph spanning the input set and having nearly optimal cost. Without any increase in the cost, in case $k = 2$, the aforementioned multigraph is efficiently transformed into a biconnected graph spanning the input point set. Furthermore, for the purpose of dynamic programming, we succeed to use a more efficient subgraph connectivity characterization in case $k = 2$ than that used in [5,6].

By using merely the aforementioned spanner transformations, we also obtain the fast randomized PTAS for finding a minimum-cost k-edge connected *multigraph* spanning a set of points in a multi-dimensional Euclidean space.

It seems unlikely that a cost-efficient transformation of a k-edge connected multigraph into a k-edge connected graph on the same point set exists. For this reason, in case of k-edge connectivity, we consider an arbitrary k-edge connected graph on the input point set instead of a spanner, and derive a series of cost-efficient transformations of the former into an r-locally-light k-edge connected graph on the input set. The transformations yield the fastest known randomized PTAS for Euclidean minimum-cost k-edge connectivity.

Our investigations of spanners with the isolation property, the *explicit* use of multigraphs instead of graphs, and the proof that nearly optimal, low cost spanners possessing the isolation property induce r-locally-light sub-multigraphs having good approximation properties are the main sources of the efficiency of the approximation schemes for Euclidean minimum-cost connectivity problems (without Steiner points) presented in this paper.

By extending the aforementioned techniques to include Steiner points, deriving the decomposition of a minimum-cost biconnected Steiner graph into minimal Steiner trees, and using the generalization of $(1+\varepsilon)$-spanner to include Steiner points called $(1+\varepsilon)$-*banyans* in [20,21], we obtain the first PTAS for minimum-cost Steiner biconnectivity and Steiner two-edge connectivity.

Organization of the paper. Section 2 provides basic terminology used in our approximation schemes. In Section 3 we outline our new PTAS for Euclidean minimum-cost biconnectivity. Section 4 sketches the PTAS for Euclidean minimum-cost k-edge connectivity in multigraphs. In Section 5 we derive our PTAS for Euclidean minimum-cost k-edge connectivity in graphs. Section 6 presents the PTAS for minimum-cost Steiner biconnectivity and Steiner two-edge connectivity. Due to space limitations most of our technical claims and their proofs are postponed to the full version of the paper.

2 Definitions

We consider geometrical graphs. A *geometrical (multi-)graph* on a set of points S in $\mathbb{R}^d$ is a weighted (multi-)graph whose set of vertices is exactly S and for which the *cost* of an edge is the Euclidean distance between its endpoints. The (total) cost of the (multi-)graph is the sum of the costs of its edges. A (multi-)graph G on S *spans* S if it is connected (i.e., there is a path in G connecting any two points in S). As in [3,5,6,20], we allow edges to deviate from straight-line segments and specify them as straight-lines

paths (i.e., paths consisting of straight-line segments) connecting the endpoints. This relaxation enables the edges to pass through some prespecified points (called *portals*) where they may be *"bent."* When all edges are straight-line segments, G is called a *straight-line graph*. For a multigraph G, the *graph induced by* G is the graph obtained by reducing the multiplicity of each edge of G to one.

We shall denote the cost of the minimum spanning tree on a point set X by $\ell(\text{MST}(X))$. A *t-spanner* of a set of points S in $\mathbb{R}^d$ is a subgraph of the complete straight-line graph on S such that for any two points $x, y \in S$ the length of the shortest path from x to y in the spanner is at most t times the Euclidean distance between x and y [1].

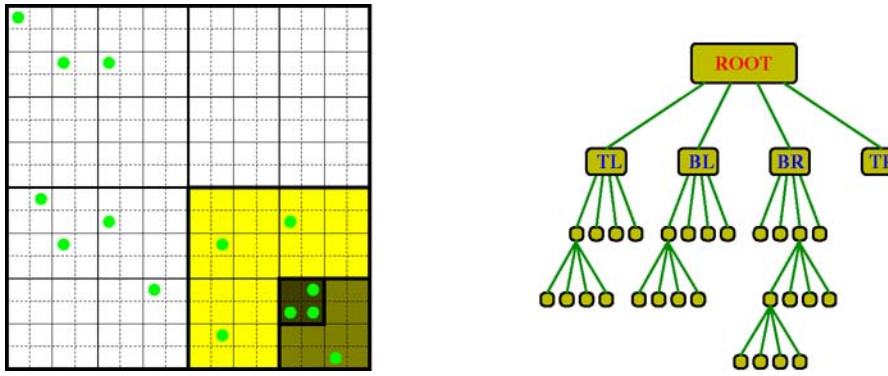


Fig. 1. Dissection of a bounding cube in $\mathbb{R}^2$ (left) and the corresponding 2^2-ary tree (right). In the tree, the children of each node are ordered from left to right: **T**op/**L**eft square, **B**ottom/**L**eft square, **B**ottom/**R**ight square, and **T**op/**R**ight square.

We hierarchically partition the space as in [3]. A *bounding box* of a set S of points in $\mathbb{R}^d$ is a smallest d-dimensional axis-parallel cube containing the points in S. A *(2^d-ary) dissection* [3] (see Figure 1) of a set of points in a cube L^d in $\mathbb{R}^d$ is the recursive partitioning of the cube into smaller sub-cubes, called *regions*. Each *region* U^d of volume > 1 is recursively partitioned into 2^d regions $(U/2)^d$. A 2^d*-ary tree* (for a given 2^d-ary dissection) is a tree whose root corresponds to L^d, and whose other non-leaf nodes correspond to the regions containing at least two points from the input set (see Figure 1). For a non-leaf node v of the tree, the nodes corresponding to the 2^d regions partitioning the region corresponding to v, are the children of v in the tree.

For any d-vector $\mathbf{a} = (a_1, \ldots, a_d)$, where all a_i are integers $0 \leq a_i \leq L$, the $\mathbf{a}$*-shifted dissection* [3,6] of a set X of points in the cube L^d in $\mathbb{R}^d$ is the dissection of the set X^* in the cube $(2\,L)^d$ in $\mathbb{R}^d$ obtained from X by transforming each point $\mathbf{x} \in X$ to $\mathbf{x} + \mathbf{a}$. A *random shifted dissection* of a set of points X in a cube L^d in $\mathbb{R}^d$ is an $\mathbf{a}$-shifted dissection of X with $\mathbf{a} = (a_1, \ldots, a_d)$ and the elements $a_1, \ldots, a_d$ chosen independently and uniformly at random from $\{0, 1, \ldots, L\}$.

A crossing of an edge with a region facet of side-length W in a dissection is called *relevant* if it has exactly one endpoint in the region and its length is at most $2\sqrt{d}\,W$. A graph is *r-gray* with respect to a shifted dissection if each facet of each region in the

dissection has at most r relevant crossings. A graph is *r-locally-light* with respect to a shifted dissection if for each region in the dissection there are at most r edges having *exactly one endpoint in the region*. A graph is *r-light* [3,20,6] with respect to a shifted dissection if for each region in the dissection there are at most r edges *crossing any of its facets*. (It is important to understand the difference between these two latter notions.)

An *m-regular* set of *portals* in a $(d-1)$-dimensional region facet N^{d-1} is an orthogonal lattice of m points in the facet where the spacing between the portals is $(N+1) \cdot m^{-1/(d-1)}$ (cf. [3]). If a graph is r-locally-light and for each facet in any region every edge crosses through one of the m portals in the facet then the graph is called *(m, r)-locally-light*.

3 Algorithm for Euclidean Biconnectivity

In this section we *sketch* a randomized algorithm that finds a biconnected graph spanning a set S of n points in $\mathbb{R}^d$ whose cost is at most $(1+\varepsilon)$ times of the minimum. We specify here only a key lemma and the structural theorem and defer the detailed description of the algorithm and its analysis to the full version of the paper.

Our algorithm starts by finding a smallest bounding box for the input point set S, rescaling the input coordinates so the bounding box is of the form $[0, \mathcal{O}(n\sqrt{d}(1+1/\varepsilon))]^d$, and moving the points in S to the closest unit grid points. We shall term the perturbed point set as *well-rounded* (see also [3,6]). Next, the algorithm finds an appropriate $(1+\Theta(\varepsilon))$-spanner of the well-rounded point set which has the so-called *(κ, c)-isolation property* for appropriate parameters κ and c [1].

Definition 3.1. *Let c, $0 < c < 1$, be a constant and let $\kappa \geq 1$. A geometrical graph G on a set of points in $\mathbb{R}^d$ satisfies the (κ, c)-isolation property, if, for each edge e of G of length l, there is a cylinder $\mathcal{C}$ of length and radius $c\,l$ and with its axis included in e such that $\mathcal{C}$ intersects at most κ edges of G.*

In the next step, the algorithm chooses a random shifted dissection and builds the corresponding shifted 2^d-ary tree for the perturbed S.

The following key lemma yields a low upper bound on the number of crossings of the boundaries of a region in the dissection by long edges of the spanner having exactly one endpoint within the region.

Lemma 3.1. *Let G be a geometrical graph spanning a set S of points in $\mathbb{R}^d$ and satisfying the (κ, c)-isolation property, where $0 < c < 1$ is a constant. There exists a constant c' such that for any region in a shifted dissection of S the number of edges of G of length at least $c'\sqrt{d}$ times the side-length of the region that have precisely one endpoint within the region is $\kappa \cdot (d/\varepsilon)^{\mathcal{O}(d)}$.*

Combining this lemma, with several lemmas that describe graph transformations reducing the number of crossings of the boundaries of a region in the dissection by short edges, we obtain the following structural theorem.

Theorem 3.1. *Let ε, λ be any positive reals and let k be any positive integer. Next, let $\mathfrak{S}$ be a $(1+\varepsilon)$-spanner for a well-rounded set S of points in $\mathbb{R}^d$ that satisfies the (κ, c)-isolation property with constant c, $0 < c < 1$, $\kappa = (d/\varepsilon)^{\mathcal{O}(d)}$, and has $n \cdot (d/\varepsilon)^{\mathcal{O}(d)}$*

edges whose total cost equals $L_{\mathfrak{S}}$. Choose a shifted dissection uniformly at random. Then one can modify $\mathfrak{S}$ to a graph $\mathcal{G}$ spanning S such that

- *$\mathcal{G}$ is r-locally-light with respect to the shifted dissection chosen, where $r = 2^{d+1} \cdot d \cdot k^2 + d \cdot (\mathcal{O}(\lambda \cdot d^{3/2}))^d + (d/\varepsilon)^{\mathcal{O}(d)}$, and*
- *there exists a k-edge connected multigraph H which is a spanning subgraph of $\mathcal{G}$ with possible parallel edges (of multiplicity at most k), whose expected (over the choice of the shifted dissection) cost is at most $(1 + \varepsilon + \frac{k \cdot L_{\mathfrak{S}}}{\lambda \cdot \ell(\mathrm{MST})})$ times cost of the minimum-cost of k-edge connected multigraph spanning S.*

Furthermore, this modification can be performed in time $\mathcal{O}(d \cdot L \cdot \log L) + n \cdot 2^{d^{\mathcal{O}(d)}} + n \cdot (d/\varepsilon)^{\mathcal{O}(d)}$, where L is the side-length of the smallest bounding box containing S.

Further, our algorithm modifies the spanner according to Theorem 3.1 producing an r-locally-light graph G where r is constant for constant ϵ and d. In the consecutive step, the algorithm runs a dynamic programming subroutine for finding a minimum-cost two-edge connected multigraph for which the induced graph is a subgraph of G (note that by Theorem 3.1 the multigraph has expected cost very close to that of the minimum-cost of a k-edge connected multigraph spanning the perturbed S). The efficiency of the subroutine relies on a new, forest-like characterization of the so called connectivity type of a multigraph within a region of the dissection. It is substantially more concise than the corresponding one used in [5,6]. Next, the algorithm transforms the multigraph to a biconnected (straight-line) graph without any increase in cost. Finally, it modifies the biconnected graph to a biconnected graph on the input set by re-perturbing its vertices.

Theorem 3.2. *The algorithm finds a biconnected graph spanning the input set of n points in $\mathbb{R}^d$ and having expected cost within $(1 + \varepsilon)$ from the optimum. The running time of the algorithm is $\mathcal{O}(n \cdot d^{3/2} \cdot \varepsilon^{-1} \cdot \log(n\, d/\varepsilon)) + n \cdot 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}}$. In particular, when d and ε are constant, then the running time is $\mathcal{O}(n \log n)$. For a constant d and arbitrary $s = \frac{1}{\varepsilon} \geq 1$ the running time is $\mathcal{O}(n\, s\, \log(n\, s) + n\, 2^{s^{\mathcal{O}(1)}})$. The algorithm can be turned into a Las Vegas one without affecting the stated asymptotic time bounds.*

Although we have used many ideas from [6] in the design of our algorithm, we have chosen the method of picking a random shifted dissection given in [3,20,21]. Therefore we can apply almost the same arguments as those used by Rao and Smith [21, Sections 2.2 and 2.3] to derandomize our algorithm at small increase of the cost

Theorem 3.3. *For every positive ε there exists a deterministic algorithm running in time $n\,(d/\varepsilon)^{\mathcal{O}(1)} \log n + n\, 2^{(d/\varepsilon)^{\mathcal{O}(d^2)}}$ that for every set of n points in $\mathbb{R}^d$ produces a biconnected graph spanning the points and having the cost within $(1+\varepsilon)$ of the minimum. In particular, when d and ε are constant, the running time is $\mathcal{O}(n \log n)$. For a constant d and arbitrary $s = \frac{1}{\varepsilon} \geq 1$ the running time is $\mathcal{O}(n\, s^{\mathcal{O}(1)} \log n + n\, 2^{s^{\mathcal{O}(1)}})$.*

4 Euclidean k-Edge Connectivity in Multigraphs

We can extend the techniques developed in the previous sections to the problem of finding a low-cost k-edge connected *multigraph* spanning a set of points in $\mathbb{R}^d$ for $k \geq 2$. To begin

with, we follow the PTAS from Section 3 up and inclusive the spanner-modification step, only changing some parameters. The resulting graph G is r-locally-light graph for $r = k^d \cdot (d/\varepsilon)^{\mathcal{O}(d^2)}$. By Theorem 3.1, there exists a k-edge connected multigraph H such that the induced graph is a subgraph of G and the expected cost of H is at most $(1+\frac{\varepsilon}{4})$ times larger than the minimum-cost k-edge connected multigraph spanning S. As was the case for the PTAS from Section 3, we can apply dynamic programming to find a minimum-cost k-edge connected multigraph H^* for which the induced graph is a subgraph of G. This time we use a more general (but less efficient) connectivity characterization from [5] yielding $2^{\mathcal{O}(r!)}$ different connectivity types. In effect, the dynamic programming is more expensive, and the total running time is $n \cdot 2^{\mathcal{O}(r!)}$. Now, it is sufficient to re-perturb the vertices of the multigraph H^* (correspondingly to the last step of the PTAS from Section 3) in order to obtain the following theorem.

Theorem 4.1. *Let k be an arbitrary positive integer. There exists an algorithm that finds a k-edge connected multigraph spanning the input set of n points in $\mathbb{R}^d$ and having expected cost within $(1+\varepsilon)$ from the optimum. The running time of the algorithm is $\mathcal{O}(n \cdot d^{3/2} \cdot \varepsilon^{-1} \cdot \log(n\, d/\varepsilon)) + n \cdot 2^{\mathcal{O}((k^d \cdot (d/\varepsilon)^{\mathcal{O}(d^2)})!)}$. In particular, when d and ε are constant, then the running time is $\mathcal{O}(n \log n) + n\, 2^{k^{\mathcal{O}(k)}}$. For a constant d and an arbitrary $s = \frac{1}{\varepsilon} \geq 1$ the running time is $\mathcal{O}(n\, s\, \log(n\, s)) + n\, 2^{(k\, s)^{\mathcal{O}(k\, s)}}$. The algorithm can be turned into a Las Vegas one without affecting the stated asymptotic time bounds.*

Recall the use of Steiner points in the first attempt of deriving PTAS for the Euclidean minimum-cost k-connectivity in [5] by allowing them solely on the approximation side. As a byproduct, we can substantially subsume the results on minimum-cost k-connectivity from [5] in the complexity aspect by using Theorem 4.1.

5 Euclidean k-Edge Connectivity in Graphs

Our spanner approach to biconnectivity relies on an efficient transformation of a two-edge connected multigraph into a biconnected graph on the same point set without any cost increase. Unfortunately, it seems that for $k > 2$ there is no any similar cost-efficient transformation between k-edge connected *multigraphs* and k-vertex- or k-edge connected *graphs*. We show in this section that an arbitrary (in particular, minimum-cost) k-edge connected graph spanning a well-rounded point set admits a series of transformations resulting in an r-locally-light k-edge connected graph on this set with a small increase in cost. By some further small cost increase, we can make the latter graph (m, r)-locally-light in order to facilitate efficient dynamic programming.

The following lemma plays a crucial role in our analysis. Intuitively, it aims at providing a similar transformation of the input graph as that presented in Lemma 3.1. The main difficulty here is in the fact that the input graph may be arbitrary (while in Lemma 3.1 we have analyzed graphs having the isolation property).

Lemma 5.1. *Let G be a spanning graph of a well-rounded point set S in $\mathbb{R}^d$. For any shifted dissection of S, G can be transformed into a graph G' satisfying the following three conditions:*

- *if G' is different from G then the total cost of G' is smaller than that of G,*

- *for any region of size W in the dissection there are positive reals x_i, $i = 1, \ldots, 2^{\mathcal{O}(d)}$, greater than $4\sqrt{d}\,W$ such that the number of edges having theirs lengths outside any of the intervals $[x_i, 2\,x_i)$, and each having precisely one endpoint within the region is $2^{\mathcal{O}(d)}$,*
- *if G is k-edge connected then so is G'.*

Lemma 5.1 guarantees that in the graph resulting from the transformation provided in the lemma no region in the dissection is crossed by too many long edges having their length outside finite number of intervals of the form $[x, 2x)$, $x > 4\sqrt{d}\,W$ and one point inside the region. The following lemma reduces the number of the remaining edges.

Lemma 5.2. *Let G be an r-gray spanning graph of a well-rounded set S of points in $\mathbb{R}^d$. Let Q be a region of size W in the dissection of S, and let x_i, $i = 1, \ldots, 2^{\mathcal{O}(d)}$, be positive reals greater than $4\sqrt{d}\,W$. If there are $\mathfrak{L}$ edges having their lengths outside the intervals $[0, 2\sqrt{d}\,W]$, $[x_i, 2\,x_i)$, $i = 1, \ldots, 2^{\mathcal{O}(d)}$, and such that each has precisely one endpoint in Q, then there are at most $\mathfrak{L} + r\,2^{\mathcal{O}(d)}$ edges crossing the facets of Q and having one endpoint in Q.*

By these two lemmas, we obtain our structure theorem for k-edge connectivity.

Theorem 5.1. *Let $\varepsilon > 0$, and let S be a well-rounded set of n points in $\mathbb{R}^d$. A minimum-cost k-edge connected graph spanning S can be transformed to a k-edge connected graph H spanning S such that*
- *H is (m, r)-locally-light with respect to the shifted dissection, where $m = (\mathcal{O}(\varepsilon^{-1} \cdot \sqrt{d} \cdot \log n))^{d-1}$ and $r = (\mathcal{O}(k^2 \cdot d^{3/2} / \varepsilon))^d$, and*
- *the expected (over the choice of the shifted dissection) cost of H is at most $(1 + \varepsilon)$ times larger than that of the minimum-cost graph.*

The concept of (m, r)-local-lightness is a very simple case of that of (m, r)-blueness used in [6]. Therefore, we can use a simplified version of the dynamic programming method of [6] involving the k-connectivity characterization from [5] in order to find an (m, r)-locally-light k-edge connected graph on a well-rounded point set satisfying the requirements of Theorem 5.1. This yields a substantially faster PTAS for the Euclidean minimum-cost k-edge connectivity than that presented in [6].

Theorem 5.2. *Let k be an arbitrary positive integer and let $\varepsilon > 0$. There exists a randomized algorithm that finds a k-edge connected graph spanning the input set of n points in $\mathbb{R}^d$ and having expected cost within $(1 + \varepsilon)$ from the optimum. The expected running time of the algorithm is $n \cdot (\log n)^{(\mathcal{O}(k^2 \cdot d^{3/2} / \varepsilon))^d} \cdot 2^{\mathcal{O}(((k^2 \cdot d^{3/2} / \varepsilon)^d)!)}$. In particular, when k, d, and ε are constant, then the running time is $n\,(\log n)^{\mathcal{O}(1)}$.*

6 Euclidean Steiner Biconnectivity

In this section we provide the first PTAS for Euclidean minimum-cost Steiner biconnectivity and Euclidean minimum-cost two-edge connectivity. For any constant dimension and ε, our scheme runs in time $\mathcal{O}(n \log n)$. Our proof relies on a decomposition of a minimum-cost biconnected Steiner graph into minimal Steiner trees and the use of the

so called $(1+\varepsilon)$-*banyans* introduced by Rao and Smith [20,21]. As a byproduct of the decomposition, we derive the first known non-trivial upper bound on the minimum number of Steiner points in an optimal solution to an n-point instance of Euclidean minimum-cost Steiner biconnectivity which is $3n-2$.

Since for any set of points X in $\mathbb{R}^d$ the minimum-cost of a biconnected graph spanning X is the same as the minimum-cost of a two-edge connected graph spanning X, in the remaining part of this section we shall focus only on the Euclidean minimum-cost Steiner biconnectivity problem. By a series of technical lemmas, we obtain the following characterization of any optimal graph solution to the Euclidean minimum-cost Steiner biconnectivity problem.

Theorem 6.1. *Let G be a minimum-cost Euclidean Steiner biconnected graph spanning a set S of $n \geq 4$ points in $\mathbb{R}^d$. Then G satisfies the following conditions:*

(i) Each vertex of G (inclusive Steiner points) is of degree either two or three.

(ii) By splitting each vertex v of G corresponding to an input point into $\deg(v)$ independent endpoints of the edges, graph G can be decomposed into a number of minimal Steiner trees.

(iii) G has at most $3n-2$ Steiner points.

6.1 PTAS

Our spanner-based method for Euclidean minimum-cost biconnectivity cannot be extended directly to include Euclidean minimum-cost Steiner biconnectivity since spanners do not include Steiner points. Nevertheless, the decomposition of an optimal Steiner solution into minimum Steiner trees given in Theorem 6.1 opens the possibility of using the aforementioned banyans to allow Steiner points for the purpose of approximating the Euclidean minimum Steiner tree problem in [20].

Definition 6.1. [20] *A $(1+\varepsilon)$-banyan of a set S of points in $\mathbb{R}^d$ is a geometrical graph on a superset of S (i.e., Steiner points are allowed) such that for any subset U of S, the cost of the shortest connected subgraph of the banyan which includes U, is at most $(1+\varepsilon)$ times larger than the minimum Steiner tree of U.*

Rao and Smith have proved the following useful result on banyans in [20].

Lemma 6.1. *Let $0 < \varepsilon < 1$. One can construct a $(1+\varepsilon)$-banyan of an n-point set in $\mathbb{R}^d$ which uses only $d^{\mathcal{O}(d^2)}\,(d/\varepsilon)^{\mathcal{O}(d)}\,n$ Steiner points and has cost within a factor of $d^{\mathcal{O}(d^2)}\,\varepsilon^{-\mathcal{O}(d)}$ of the minimum Steiner tree of the set. The running time of the construction is $d^{\mathcal{O}(d^2)}\,(d/\varepsilon)^{\mathcal{O}(d)}\,n + \mathcal{O}(d\,n\log n)$.*

By combining the definition of $(1+\varepsilon)$-banyan with Theorem 6.1 (2) and Lemma 6.1, we get the following lemma.

Lemma 6.2. *For a finite point set S in $\mathbb{R}^d$ let $\mathfrak{B}$ be a $(1+\varepsilon/4)$-banyan constructed according to Lemma 6.1. Let $\mathfrak{B}_m$ be the multigraph obtained from $\mathfrak{B}$ by doubling its edges. There is a two-edge connected sub-multigraph of $\mathfrak{B}_m$ which includes S and whose cost is within $(1+\varepsilon/4)$ of the minimum cost of two-edge connected multigraph on any superset of S.*

Let $\mathfrak{B}_m^*$ be the multigraph resulting from scaling and perturbing all the vertices (i.e., also the Steiner points) of the multigraph $\mathfrak{B}_m$ specified in Lemma 6.2 according to the first step in Section 3. The vertices of $\mathfrak{B}_m^*$ are on a unit grid $[0, L]^d$ and, by arguing analogously as in Section 3, a minimum-cost two-edge connected sub-multigraph of $\mathfrak{B}_m^*$ that includes S is within $(1 + \varepsilon/4)$ of a minimum-cost two-edge connected sub-multigraph of $\mathfrak{B}_m$ that includes S.

The patching method of [5] applied to $\mathfrak{B}_m^*$ yields the following structure theorem.

Theorem 6.2. *Choose a shifted dissection of the set of vertices of the banyan $\mathfrak{B}$ at random. Then $\mathfrak{B}_m^*$ can be modified to a multigraph $\mathfrak{B}_m'$ such that:*

- *$\mathfrak{B}_m'$ is r-light with respect to the shifted dissection, where $r = (\mathcal{O}(\sqrt{d}/\varepsilon))^{d-1}$,*
- *the set of vertices of $\mathfrak{B}_m'$ includes that of $\mathfrak{B}_m^*$ and some additional vertices placed at the crossings between the edges of $\mathfrak{B}_m^*$ and the boundaries of the regions in the shifted dissection,*
- *there exists a two-edge connected sub-multigraph of $\mathfrak{B}_m'$ including S whose expected cost is within $(1+\varepsilon/4)$ of the minimum-cost of two-edge connected sub-multigraph of $\mathfrak{B}_m^*$ that includes S.*

To find such a subgraph of $\mathfrak{B}_m'$ efficiently we apply a simplification of the dynamic programming method used by the PTAS from Section 3 to the set of vertices of $\mathfrak{B}_m^*$ (it would be even simpler to use a modification of the dynamic programming approach from [5]). In effect we can find $\mathfrak{B}_m'$ in expected time $\mathcal{O}(n \log n)$ for constant ε and d. By combining this with Lemma 6.2, Theorem 6.2, and the efficient transformation of two-edge connected multigraphs into biconnected graphs, we obtain the main result in this section.

Theorem 6.3. *There exists an approximation algorithm for the minimum-cost Steiner biconnectivity (and two-edge connectivity) which for any $\varepsilon > 0$ returns a Euclidean Steiner biconnected (or two-edge connected) graph spanning the input set of n points in $\mathbb{R}^d$ and having expected cost within $(1 + \varepsilon)$ from the optimum. The running time of the algorithm is $\mathcal{O}(n\, d^{3/2}\, \varepsilon^{-1} \log(n\, d/\varepsilon)) + d^{\mathcal{O}(d^2)}\, (d/\varepsilon)^{\mathcal{O}(d)}\, n + n\, 2^{(\sqrt{d}/\varepsilon)^{d\,(\mathcal{O}(\sqrt{d}/\varepsilon))^{d-1}}}$. In particular, when d and ε are constant, then the running time is $\mathcal{O}(n \log n)$. The algorithm can be turned into a Las Vegas one without affecting asymptotic time bounds.*

Acknowledgments

We thank unknown referees for their valuable comments and suggestions (because of space considerations not all of the suggestions could be implemented).

References

1. S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: Short, thin, and lanky. *Proc. 27th ACM STOC*, pp. 489–498, 1995.
2. R. K. Ahuja, T. L. Magnanti, J. B. Orlin, and M. R. Reddy. Applications of network optimization. In Nemhauser, editors, *Handbooks in Operations Research and Management Science*, volume 7: *Network Models*, ch. 1, pp. 1–83. North-Holland, Amsterdam, 1995.

3. S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. Assoc. Comput. Mach.*, 45(5):753–782, 1998.
4. D. Bienstock, E. F. Brickell, and C. L. Monma. On the structure of minimum-weight k-connected spanning networks. *SIAM J. Disc. Math.*, 3(3):320–329, 1990.
5. A. Czumaj and A. Lingas. A polynomial time approximation scheme for Euclidean minimum cost k-connectivity. *Proc. 25th ICALP*, pp. 682–694, 1998.
6. A. Czumaj and A. Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. *Proc. 10th ACM-SIAM SODA*, pp. 281–290, 1999.
7. G. N. Frederickson and J. JáJá. On the relationship between the biconnectivity augmentation and Traveling Salesman Problem. *Theoret. Comput. Sci.*, 19(2):189–201, 1982.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-completeness*. Freeman, New York, NY, 1979.
9. M. Grötschel, C. L. Monma, and M. Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40(2):309–330, 1992.
10. M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. *Handbooks in Operations Research and Management Science*, volume 7: *Network Models*, ch. 10, pp. 617–672. North-Holland, Amsterdam, 1995.
11. E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM J. Appl. Math.*, 16(1):1–29, 1968.
12. M. X. Goemans and D. P. Williamson. The primal-dual method for approximation algorithms and its application to network design problems. Chapter 4 in [15], pp. 144–191.
13. J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. To appear in *Proc. 7th SWAT*, 2000.
14. D. F. Hsu and X-D. Hu. On short two-connected Steiner networks with Euclidean distance. *Networks*, 32(2):133–140, 1998.
15. D. S. Hochbaum, editor. *Approximation Algorithms for $\mathcal{NP}$-Hard Problems*. PWS Publishing Company, Boston, MA, 1996.
16. F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, Amsterdam, 1992.
17. S. Khuller. Approximation algorithms for finding highly connected subgraphs. Chapter 6 in [15], pp. 236–265.
18. D. Krznaric, C. Levcopoulos, G. Narasimhan, B. J. Nilsson, and M. Smid. The minimum 2-connected subgraph of a set of points. Manuscript, September 15, 1997.
19. J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999.
20. S. B. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans." *Proc. 30th ACM STOC*, pp. 540–550, 1998. A full version appeared in [21].
21. S. B. Rao and W. D. Smith. Improved approximation schemes for geometrical graphs via "spanners" and "banyans." May 1998. An extended abstract appeared in [20].
22. M. Stoer. *Design of Survivable Networks*, volume 1531 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1992.
23. K. Steiglitz, P. Weiner, and D. J. Kleitman. The design of minimum cost survivable networks. *IEEE Trans. Circuit Theor.*, 16:455–460, 1969.

Approximate TSP in Graphs with Forbidden Minors

Michelangelo Grigni *

Dept. of Mathematics and Computer Science,
Emory University, Atlanta Georgia 30322, USA,
mic@mathcs.emory.edu

Abstract. Given as input an edge-weighted graph, we analyze two algorithms for finding subgraphs with low total edge weight. The first algorithm finds a separator subgraph with a small number of components, and is analyzed for graphs with an arbitrary excluded minor. The second algorithm finds a spanner with small stretch factor, and is analyzed for graphs in a hereditary family $\mathcal{G}(k)$. These results imply (i) a QPTAS (quasi-polynomial time approximation scheme) for the TSP (traveling salesperson problem) in unweighted graphs with an excluded minor, and (ii) a QPTAS for the TSP in weighted graphs with bounded genus.

Keywords: graph minor, genus, separator, spanner, TSP, approximation scheme.

1 Introduction

In the *traveling salesperson problem* (TSP) we are given n sites and their distance matrix, and our goal is to find a simple closed tour of the sites with minimum total distance. The TSP has driven both practical and theoretical algorithm research for several decades [9]. Most variants are NP-hard, and therefore much attention is given to approximate solutions for *metric TSP*, where the distance matrix is a metric (nonnegative, symmetric, and satisfying the triangle inequality). An algorithm of Christofides [6] finds a metric TSP solution with cost at most 3/2 times optimal. We would prefer a polynomial time approximation scheme (PTAS); that is, for each $\varepsilon > 0$, we would like a polynomial time algorithm which produces a solution with cost at most $1+\varepsilon$ times optimal. However, metric TSP is MAXSNP-hard even when all distances are one or two [12], and so there is some positive ε_0 such that finding a $1+\varepsilon_0$ approximation is NP-hard. Indeed, the 3/2 guarantee of Christofides has not been improved (although in practice, other heuristics are much better).

However, there has been recent progress in certain restricted metric spaces. In [8] we found a PTAS for the TSP on the nodes of an unweighted planar graph, where the metric is given by shortest path lengths in the graph. We later generalized [4] this to allow distances defined by non-negative edge costs.

* Supported by NSF Grant number CCR-9820931.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 869–877, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Arora [3] (also Mitchell [11]) gave a PTAS for the TSP and related problems for points in a Euclidean space of fixed dimension. Roughly speaking, all of these results depend on the ability to find inexpensive and "well-connected" separators.

In this paper we extend the methods of [4] from planar graphs to larger graph families. This leads us to a general notion of well-connected separator that may have other algorithmic applications. We present two subgraph finding algorithms; in both algorithms the goal is a *light* subgraph, meaning that it has low total edge weight. In Section 3 we give an algorithm finding a light separator subgraph with few connected components, in graphs from any family with a nontrivial excluded minor. In Section 4 we give an algorithm finding a light spanner with low stretch factor, in graphs from a family $\mathcal{G}(k)$, to be defined. This family includes graphs of bounded genus.

Finally, in Section 5 we sketch a QPTAS (quasi-polynomial time approximation scheme) for metric TSP in two situations. First, for the shortest path metric in an unweighted graph from a family with an excluded minor. Second, for the shortest path metric in an edge-weighted graph from family $\mathcal{G}(k)$, for any fixed k. Both schemes run in time $n^{O(\log \log n)}$.

2 Preliminaries

All graphs in this paper are undirected and simple. A graph $G = (V, E)$ is *edge-weighted* if it has a non-negative weight (or length) $\ell(e)$ on each edge $e \in E$; it is *vertex-weighted* if it has a non-negative weight $w(v)$ on each vertex $v \in V$. A subgraph H of G inherits these weights on its edges and vertices. The total edge weight and vertex weight in H are denoted by $\ell(H)$ and $w(H)$, respectively. The number of connected components in H is denoted $\#(H)$.

When G is edge-weighted, $d_G(u, v)$ denotes the minimum length $\ell(P)$ of a path P connecting endpoints u and v. This is zero when $u = v$, and $+\infty$ when u and v are disconnected. $G' = (V, E')$ is a *spanning subgraph* if it spans each component of G. Clearly $d_G(u, v) \leq d_{G'}(u, v)$; the *stretch factor* $\mathrm{sf}(G', G)$ is the minimum s such that $d_{G'}(u, v) \leq s \cdot d_G(u, v)$ for all $u, v \in V$ (it suffices to consider only edge pairs $\{u, v\} \in E$). When $\mathrm{sf}(G', G) \leq s$, we say that G' is an *s-spanner* in G.

Given a graph G, a *minor* of G is a graph resulting from some sequence of edge deletion, vertex deletion, or edge contraction operations (denoted $G - e$, $G - v$, and G/e respectively). Since we only consider simple graphs, we discard self-loops and parallel edges. We say G has an H-minor (denoted $H < G$) if G has a minor isomorphic to H. A *hereditary graph property* is a class $\mathcal{P}$ of graphs closed under isomorphism, such that whenever G is in $\mathcal{P}$, so are its minors. In a series of papers, Robertson and Seymour show that every hereditary graph property is characterized by a finite set of forbidden minors (see [7] for an overview). The prime example is Kuratowski's characterization of planar graphs by the forbidden minors $\{K_5, K_{3,3}\}$.

For a subset X of vertices in G, an X-flap is the vertex set of a connected component of $G-X$. Given a vertex-weighted graph G, a *separator* is a subgraph S such that every $V(S)$-flap has weight at most $w(G)/2$. Note that separators are usually defined as just a vertex set, but in this paper we are interested in a tradeoff between $\ell(S)$ and $\#(S)$.

Let $V(G)^{\leq k}$ denote the collection of sets of at most k vertices in G. A *haven of order* k is a function β assigning an X-flap to each $X \in V(G)^{\leq k}$, such that $\beta(Y) \subseteq \beta(X)$ whenever $X \subseteq Y \in V(G)^{\leq k}$. Given a vertex-weighted graph G and a non-separator vertex subset X, let $\beta^w(X)$ denote the unique X-flap with weight exceeding $w(G)/2$. If X is a separator, let $\beta^w(X) = \emptyset$. Note that if G has no separator of size k, then β^w (restricted to $V(G)^{\leq k}$) is a haven of order k.

3 A Well-Connected Separator

Alon, Seymour, and Thomas [1] give a polynomial time algorithm to find a separator in a graph with an excluded minor. Specifically, given as input a vertex-weighted graph G and a graph H, their algorithm either finds an H-minor in G, or it finds a separator in G with at most $h^{3/2}n^{1/2}$ vertices, where h is the number of vertices in H. In particular, if we fix a non-trivial hereditary graph property $\mathcal{P}$ and only consider inputs $G \in \mathcal{P}$, then this algorithm finds separators of size $O(n^{1/2})$; this generalizes the planar separator theorem [10].

They (and we) only consider the case $H = K_h$, since a K_h-minor implies an H-minor. A *covey* is a forest $\mathcal{C}$ in G such that each pair of component trees is connected by an edge of G. A covey with $\#(\mathcal{C}) = h$ witnesses a K_h-minor.

In our application, G is also edge-weighted. We modify their algorithm to allow a trade-off between the total edge weight of the separator and the number of its connected components. We claim the following:

Theorem 1. *There is a polynomial time algorithm taking as input a vertex-weighted edge-weighted graph G, a positive integer h, and a positive real $\varepsilon \leq 1$, and which produces as output either:*
(a) a K_h-minor in G, or
(b) a separator S of G such that $\ell(S) \leq \varepsilon\, h\, \ell(G)$ and $\#(S) \leq h^2/\varepsilon$.

We use much of their algorithm unchanged, so in this abstract we simply describe and analyze our changes. Our basic subroutine is the following slight modification of [1, Lemma 2.1]:

Lemma 2. *Let G be an edge-weighted graph with m edges, let $A_1, \ldots, A_k$ be subsets of $V(G)$, and let ε be a positive real number. There is a polynomial time algorithm which returns either:*
(i) a tree T in G such that $\ell(T) \leq \varepsilon\, \ell(G)$ and $V(T) \cap A_i \neq \emptyset$ for $i = 1, \ldots, k$; or
(ii) a set $Z \subseteq V(G)$ such that $|Z| \leq (k-1)/\varepsilon$ and no Z-flap intersects all of $A_1, \ldots, A_k$.

The proof of this lemma is essentially the same, except that we use a shortest paths tree rather than breadth first search. The rest of the proof is omitted.

The algorithm is iterative. After t steps of the algorithm, we have a subgraph X_t and a covey $\mathcal{C}_t$; initially X_0 and $\mathcal{C}_0$ are empty. In step t of the algorithm, we halt if either $\#(\mathcal{C}_{t-1}) \geq h$ or X_{t-1} is a separator. Otherwise, we let $B_{t-1} = \beta^w(X_{t-1})$ and we invoke Lemma 2 on $G[B_{t-1}]$, where the A_i's are the neighborhoods of the component trees in $\mathcal{C}_{t-1}$; we call this step either a T-step or a Z-step, depending on which is returned. The returned T or Z is then used to define X_t and $\mathcal{C}_t$, according to several cases as described in [1]. We have these invariants:

1. X_t is a subgraph of $\mathcal{C}_t$.
2. For each component tree C in $\mathcal{C}_t$, either $X_t \cap C$ equals some T returned in a T-step, or $X_t \cap C$ is a set of disconnected vertices contained in some Z returned in a Z-step.
3. $B_t \subseteq B_{t-1}$; and if these are equal, then $X_t \subset X_{t-1}$.
4. $V(\mathcal{C}_t)$ and B_t are disjoint.

By the first invariant, X_t is the union of at most h parts of the form $X_t \cap C$. By the second invariant and Lemma 2, each part has $\ell(X_t \cap C) \leq \varepsilon \cdot \ell(G)$ and $\#(X_t \cap C) \leq (k-1)/\varepsilon$. Therefore $\ell(X_t) \leq h\ \varepsilon\ \ell(G)$ and $\#(X_t) \leq h^2/\varepsilon$, as required in Theorem 1(b).

By invariant 3 above, we see that the sequence of pairs $(|B_t|, |X_t|)$ is lexicographically decreasing; therefore the algorithm halts after at most n^2 iterations. In fact an improved time analysis is possible, but we omit it here.

Remark. Our algorithm (and the original) may also be useful in situations where we have a G with no K_h-minor, but the vertex-weighting w is unknown. Observe that w affects the algorithm in only two ways. First, it can tell us when to stop because X_t is a separator. Second, when we update X_{t-1}, B_{t-1} splits into disjoint flaps, and w tells us which flap to take as the next B_t. Since the B_t's decrease, the tree of possible computations (depending on w) has at most n leaves. The tree depth is at most the maximum number of iterations, considered above. Therefore there is a polynomial size collection of vertex sets in G, such that for any weighing w, one of them is a separator satisfying the conditions of Theorem 1(b).

4 The Span Algorithm

Althöfer *et al.* [2] introduced the following greedy algorithm to find an s-spanner in an edge-weighted graph G. The parameter s is at least one, and $d_{G'}(e)$ denotes the length of the shortest path in G' connecting the endpoints of edge e (the length may be $+\infty$):

```
Span(G = (V, E), s):
    G' ← (V, ∅)
    for all e ∈ E in non-decreasing ℓ order do
        if s · ℓ(e) < d_{G'}(e) then add e to G'
    return G'
```

In the resulting G', we have $d_{G'}(e) \leq s \cdot \ell(e)$ for every edge $e \in E$; therefore G' is an s-spanner. By comparison with Kruskal's algorithm, we see that $T(G) \stackrel{\text{def}}{=} \mathrm{Span}(G, n-1)$ is a minimum spanning forest in G, and $\mathrm{Span}(G, s)$ always contains $T(G)$. The Span algorithm is idempotent in this sense: if $G' = \mathrm{Span}(G, s)$, then $G' = \mathrm{Span}(G', s)$.

Define the *tree weight* of G' as $\mathrm{tw}(G') = \ell(G')/\ell(T(G'))$ (note that $T(G') = T(G)$). We seek a tradeoff between $\mathrm{sf}(G', G)$ and $\mathrm{tw}(G')$. The algorithm has two extremes: $\mathrm{Span}(G, n-1)$ has tree weight one but may have stretch factor $n-1$; $\mathrm{Span}(G, 1)$ has stretch factor one but may have tree weight nearly $n^2/4$. For intermediate s, the following tradeoff is known [2, Thm. 2]:

Theorem 3. *If $s > 1$ and G is planar then* $\mathrm{tw}(\mathrm{Span}(G, s)) \leq 1 + 2/(s-1)$.

With s close to one, this theorem is a critical element of the approximation scheme for the TSP in weighted planar graphs [5]. Motivated by this application, we seek to extend the result to larger graph families.

Definition 4. *Suppose G is a graph, ℓ is an edge weighting in G, and T is a spanning forest. Define:*

1. $\mathrm{gap}_\ell(e) = d_{G-e}(e) - \ell(e)$.
2. $\mathrm{gap}(G, T) = \max_\ell(\sum_{e \notin T} \mathrm{gap}_\ell(e))/\ell(T)$, *where ℓ ranges over all edge weightings such that $T = T(G)$.*
3. $\mathrm{gap}(G) = \max_T \mathrm{gap}(G, T)$, *where T ranges over all spanning forests.*
4. *the graph class $\mathcal{G}(k) = \{G |\ \mathrm{gap}(G) \leq k\}$.*

Remark. Given G and T, $\mathrm{gap}(G, T)$ is the value of a linear program; suppose ℓ achieves the maximum. If e is a cut edge then $\mathrm{gap}_\ell(e)$ is infinite, but it does not matter since $e \in T$ (and we may set $\ell(e) = 0$). For other edges e we may assume $\mathrm{gap}_\ell(e) \geq 0$, because otherwise we could improve ℓ by setting $\ell(e) = d_{G-e}(e)$.

Theorem 5. *$\mathcal{G}(k)$ is a hereditary graph property.*

Proof. $\mathcal{G}(k)$ is easily closed under isomorphism; we need to show $\mathrm{gap}(H) \leq \mathrm{gap}(G)$ whenever $H < G$. Take ℓ and T' in H such that $\mathrm{gap}(H, T') = \mathrm{gap}(H)$. In G we will define an edge weighting (also denoted ℓ) and a spanning forest T. By induction it suffices to consider these three cases:
Case $H = G - e$: If e connects two components of $G - e$, let $\ell(e) = 0$ and include e in T so $T - e = T'$. Otherwise let $\ell(e) = d_{G-e}(e)$ and $T = T'$.
Case $H = G - v$: By deleting edges first, we may assume v is isolated. Then ℓ is unchanged, and we add the isolated v to T.
Case $H = G/e$: By deleting edges first, we may assume that no two edges in G merge to one in G/e. Let $\ell(e) = 0$ and include e in T so that $T/e = T'$.

In all cases we have constructed ℓ and T such that $(\sum_{e \notin T} \mathrm{gap}_\ell(e))/\ell(T) = \mathrm{gap}(H, T')$, therefore $\mathrm{gap}(G) \geq \mathrm{gap}(H)$ as required. □

Let $g(H)$ denote the girth of a graph H. By considering the uniform edge weighting $\ell \equiv 1$, we have:

Corollary 6. $\mathrm{gap}(G) \geq \max_H (g(H) - 2)(|E(H)|/(|V(H)| - 1) - 1)$, *where H ranges over all 2-connected minors of G.*

We now relate $\mathrm{gap}(G)$ to the Span algorithm. Let $\mathrm{gap}'_\ell(e)$ denote the edge gap in G', that is $d_{G'-e}(e) - \ell(e)$.

Lemma 7. *If $G' = \mathrm{Span}(G, s)$, then $\ell(e) < 1/(s-1) \cdot \mathrm{gap}'_\ell(e)$ for all e in G'.*

Proof. We follow [2, Lem. 3]. Let P be the shortest path in $G' - e$ connecting the endpoints of e. Just before the algorithm inserts the longest edge $f \in \{e\} \cup P$, we have $s \cdot \ell(e) \leq s \cdot \ell(f) < d_{G'-f}(f) \leq \ell(P) = d_{G'-e}(e)$. □

Theorem 8. *If $s > 1$ and $G \in \mathcal{G}(k)$, then* $\mathrm{tw}(\mathrm{Span}(G, s)) \leq 1 + k/(s-1)$.

Proof. We are given G with some weighting ℓ. Let $G' = \mathrm{Span}(G, s)$; we need to show $\mathrm{tw}(G') \leq 1 + k/(s-1)$. Theorem 5 implies $\mathrm{gap}(G') \leq k$. Let $T = T(G')$. By the definition of $\mathrm{gap}(G')$, $\sum_{e \notin T} \mathrm{gap}'_\ell(e) \leq k\ell(T)$. By the lemma we have $\sum_{e \notin T} \ell(e) \leq k/(s-1)\ell(T)$, and the result now follows. □

Remark. Theorem 3 is proved by showing $\mathcal{G}(2)$ contains all planar graphs. Although not stated in this way, they construct a feasible point in a linear program dual to the definition of $\mathrm{gap}(G, T)$ (it is feasible even if we drop the $T = T(G)$ constraint).

Lemma 6 implies that K_h is a forbidden minor in $\mathcal{G}(h/2 - 1 - \varepsilon)$; we conjecture a converse relation.

Conjecture 9. There is a function $f(\cdot)$ such that $\mathcal{G}(f(h))$ contains all graphs with no K_h-minor.

Absent this conjecture, we offer weaker evidence that $\mathcal{G}(k)$ is interesting:

Lemma 10. *Suppose G has genus g; that is, G may be drawn without crossings on an orientable surface with g handles. Then $G \in \mathcal{G}(12g - 4)$.*

Proof. (Sketch.) Suppose G is drawn in an orientable surface with g handles. Choose a spanning tree T such that $\mathrm{gap}(G, T) = \mathrm{gap}(G)$. For edges $e, f \notin T$, say they are equivalent if the cycles in $e + T$ and $f + T$ are homotopic. If we take T and all the edges of one equivalence class, we get a planar subgraph of G. Suppose there are h equivalence classes; then G is the union of h planar subgraphs $G_1, \ldots, G_h$ with a common spanning tree T. By Definition 4 we see $\mathrm{gap}(G, T) \leq \sum_{i=1}^{h} \mathrm{gap}(G_i, T)$, and this is at most $2h$.

It now suffices to show $h \leq 6g - 2$. We contract T to a point p, so the arcs become non-crossing non-homotopic loops based at p. We pick one loop of each class, and consider the faces defined by these h loops. We may assume that each face is a 2-cell, otherwise we could add another loop. Since no two loops are homotopic, no face has two sides. There may be one face bounded by one loop e, but then the other side of e has at least four sides; all other faces have at least three sides. Therefore $2e = \sum_\Delta |\Delta| \geq 3f - 1$, where e is the number of loops, f is the number of faces, and $|\Delta|$ is the number of sides of face Δ. Combining this with Euler's formula $v - e + f = 2 - 2g$ gives our bound (here $v = 1$). A simple construction shows $h = 6g - 2$ is achievable for $g \geq 1$. □

5 The Approximation Schemes

We will reuse the methods introduced for planar graphs [8,4], and only sketch them here. We are given as input a connected graph G, and a parameter $\varepsilon > 0$. Our goal is to find a circuit in the graph visiting each vertex at least once, and with length within $1 + \varepsilon$ times the minimum (this is equivalent to the original metric TSP formulation). The minimum lies between $\ell(T(G))$ and $2\ell(T(G))$, so it suffices to find a solution with additive error at most $\varepsilon\ell(T(G))$. We need to handle these two cases:

Case G is unweighted and has no K_h-minor: We introduce the uniform edge weighting $\ell \equiv 1$. By Mader's Theorem [7, 8.1.1] there is a constant K such that $\ell(G) \leq K\ell(T(G))$ (the best K is $\Theta(h\sqrt{\log h})$ [14]).

Case G is weighted and in $\mathcal{G}(k)$: We replace G by $\mathrm{Span}(G, 1 + \varepsilon/4)$; this substitution introduces at most $(\varepsilon/2)\ell(T(G))$ additive error. Theorem 8 implies $\ell(G) \leq K\ell(T(G))$, where $K = 1 + 4k/\varepsilon$. Also by Lemma 6 we know G contains no K_h-minor, for $h \geq 2(k + 1)$.

Now in either case we know that G has no K_h-minor, and that $\ell(G) \leq K\ell(T)$. We now need to find a circuit within $(\varepsilon/2)\ell(T(G))$ of optimal in time $n^{O(\log\log n)}$, where the hidden constant depends on ε, h, and K.

Given a separator S of G, it is easy to find a *separation*: that is a triple (S, A_1, A_2) such that S is a subgraph, $A_1 \cup A_2 = V(G)$, $A_1 \cap A_2 = V(S)$, there are no edges between $A_1 - S$ and $A_2 - S$, and each $A_i - S$ has vertex weight at most $(2/3)w(G)$. So by Theorem 1, we have:

Corollary 11. *Suppose G is an edge-weighted graph with $\ell(G) \leq \ell(T(G))$ and no K_h-minor, and $\delta \leq 1$ is a positive real number. Then there is a polynomial time algorithm finding a separation (S, A_1, A_2) of G such that $\ell(S) \leq \delta\ \ell(G)$ and $\#(S) \leq h^3/\delta$.*

We give G a uniform vertex-weighting w. We will build a linear size decomposition tree $\mathcal{T}$ of G, by repeated application of Corollary 11 with the parameter $\delta = \gamma\varepsilon/\log n$, where $\gamma > 0$ is a constant to be determined.

If a weighted graph F in $\mathcal{T}$ has less than $\Theta(\delta^{-2})$ vertices, then it is a leaf. Otherwise, we apply Corollary 11 to find a separation (S_F, A_1, A_2) in F. For each A_i we let F_i denote the graph that results from $F[A_i]$ by contracting each component of S_F to a point; F_1 and F_2 are the children of F in $\mathcal{T}$. We call the new contracted points *portal points*, and give them (for now) zero weight. Note that F_i may also inherit portal points from F, and that each edge of F appears in at most one child.

Since $w(F_i) \leq 2/3w(F)$, the depth of $\mathcal{T}$ is $O(\log n)$. We introduce at most $f = h^3/\delta$ new portals in each split, so every graph in $\mathcal{T}$ has at most p portals, where $p = O(f \log n) = O(h^3(\log^2 n)/\varepsilon)$. Since each original edge of G appears at most once in a level of $\mathcal{T}$, the edges of all S_F contracted in a single level have total weight at most $\delta\ \ell(G)$. Summing over all levels, the total weight of all the contracted spanners is $O(\varepsilon\ell(G))$. By a suitable choice of $\gamma = \Theta(1/K)$, we may ensure that this is at most $(\varepsilon/4)\ell(T(G))$.

Consider the optimum circuit τ in G. After performing the splits and contractions, τ has an image τ_F in each graph F of $\mathcal{T}$; τ_F enters and leaves F through its portals in some order, defining a sequence of portal-terminated paths within F, covering its vertices. Furthermore, by a simple patching argument [4, Lemma 3.2], we may rearrange τ (without changing its cost) so that each τ_F uses each portal of F at most twice as an endpoint.

Therefore, we are led to the following problem, which we will solve approximately by dynamic programming in the tree $\mathcal{T}$. Given a graph $F \in \mathcal{T}$ and an sequence σ of its portals where each portal appears at most twice in σ, a σ-tour of F is a sequence of paths covering F, with path endpoints as specified by the list σ. For each σ, we want to find a near-optimal σ-tour in F.

If F is a leaf in $\mathcal{T}$, then we exactly solve each such problem in $2^{O(1/\delta)}$ time, using the ordinary minor separator theorem [1]. If F has children, then after we have solved all such problems for F_1 and F_2, we may solve them for F as follows. Consider all pairings of a σ_1-tour in F_1 and a σ_2-tour in F_2; if they are compatible, their paths patch together to give us some σ-tour in F. For each σ, we record the cheapest combination obtained; we then recover a true σ-tour in F by "uncontracting" the edges of S_F and charging each uncontracted edge at most twice.

As shown above, the total weight of all these charged edges (over all of $\mathcal{T}$) is at most $(\varepsilon/4)\ell(T(G))$, therefore the total additive error in these contributed by this uncontraction is $(\varepsilon/2)\ell(T(G))$. We can show that this is the only source of additive error in our solution, so we have the promised approximation scheme.

The time of the above algorithm is roughly the number of dynamic programming subproblems, which is $n^{O(1)}p^{O(p)}$. With our previous bound for p, this is $n^{O((Kh^3/\varepsilon)\log n \log\log n)}$. In fact we can do better; by using the portal weighing scheme of [8], we can ensure that each graph in $\mathcal{T}$ has at most $p = 6f$ portals, while $\mathcal{T}$ still has $O(\log n)$ depth. With this improvement, our time bound is $n^{O((Kh^3/\varepsilon)\log\log n)}$.

6 Open Problems

Of course proving Conjecture 9 would help unify our present results.

We would prefer a true polynomial time approximation scheme, rather than quasi-polynomial. Our obstacle is the number of portal orderings σ that we must consider for each F. In the case of planar graphs [4], we overcame the obstacle by observing that we only needed to consider those σ-tours corresponding to non-crossing paths in an embedding of F on a sphere with $O(1)$ holes. This observation reduces the number of σ considered to a simple exponential $2^{O(p)}$, and consequently the total time is polynomial in n. In the present situation, a bound like the above is unknown, but it is at least plausible. This is because graphs with a forbidden minor are characterized [13] in terms of blocks that can be "nearly drawn" on a 2-manifold with a bounded genus and number of holes.

We would also like to solve the Steiner version of the problem, where along with G we are given a set of "terminal" vertices, and we want to find a minimum

length tour visiting all the terminals. The remark at the end of Section 3 is a preliminary step in that direction.

Acknowledgment

We thank Robin Thomas for help with Lemma 10.

References

1. N. Alon, P. D. Seymour, and R. Thomas. A separator theorem for graphs with an excluded minor and its applications. In *Proc. 22nd Symp. Theory of Computing*, pages 293–299. Assoc. Comput. Mach., 1990.
2. I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9(1):81–100, 1993. An early version appeared in SWAT'90, LNCS V. 447.
3. S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, Sept. 1998.
4. S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 33–41, San Francisco, California, 25–27 Jan. 1998.
5. S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *9th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 33–41, Jan. 1998.
6. N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. In J. F. Traub, editor, *Symposium on New Directions and Recent Results in Algorithms and Complexity*, page 441, NY, 1976. Academic Press. Also CMU Tech. Report CS-93-13, 1976.
7. R. Diestel. *Graph theory.* Springer-Verlag, New York, 1997. Translated from the 1996 German original.
8. M. Grigni, E. Koutsoupias, and C. Papadimitriou. An approximation scheme for planar graph TSP. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)*, pages 640–645, Los Alamitos, Oct. 1995. IEEE Computer Society Press.
9. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem.* Wiley, 1985, 1992.
10. R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.
11. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SICOMP: SIAM Journal on Computing*, 28, 1999.
12. C. H. Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18:1–11, 1993.
13. N. Robertson and P. D. Seymour. Graph minors XVII: Excluding a non-planar graph. Submitted.
14. A. Thomason. An extremal function for contractions of graphs. *Math. Proc. Cambridge Philos. Soc.*, 95(2):261–265, 1984.

Polynomial Time Approximation Schemes for General Multiprocessor Job Shop Scheduling

Klaus Jansen[1] and Lorant Porkolab[2]

[1] Institut für Informatik und praktische Mathematik, Christian-Albrechts-Universität zu Kiel, 24 098 Kiel, Germany, kj@informatik.uni-kiel.de

[2] Department of Computing, Imperial College of Science, Technology and Medicine, London SW7 2BZ, United Kingdom, porkolab@doc.ic.ac.uk

Abstract. We study preemptive and non-preemptive versions of the general multiprocessor job shop scheduling problem: Given a set of n tasks each consisting of at most μ ordered operations that can be processed on different (possibly all) subsets of m machines with different processing times, compute a schedule with minimum makespan where operations belonging to the same task have to be scheduled according to the specified order. We propose algorithms for this problem that compute approximate solutions of any positive ϵ accuracy and run in $O(n)$ time for any fixed values of m, μ and ϵ. These results include (as special cases) many recent developments on polynomial time approximation schemes for scheduling jobs on unrelated machines [12], multiprocessor tasks [5,11,13], and classical open, flow and job shops [14,15].

1 Introduction

We consider the multiprocessor job shop scheduling problem: Given a set of n independent tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$ and a set of m machines $M = \{1, \ldots, m\}$. Each task T_j consists of $\mu_j \leq \mu$ multiprocessor operations $O_{j1}, \ldots, O_{j\mu_j}$ that have to be scheduled in the given order. For each operation O_{ji} there is a set m_{ji} consisting of at most 2^m different modes, where each processing *mode* $\rho \in m_{ji}$ corresponds to a non-empty subset $\rho \subseteq M$ of processors and specifies the operation's execution time $p_{ji}(\rho)$ on that particular processor set. The objective is to minimize the makespan, i.e. the maximum completion time over all feasible schedules.

We focus here on the large and non-trivial (NP-hard) subclass of the problem, where both m and μ are fixed. Following the standard notation scheme of the scheduling literature (see e.g. [16]), the non-preemptive (preemptive) version of the latter problem is denoted by $Jm|set_{ij}, op \leq \mu|C_{max}$ ($Jm|set_{ij}, op \leq \mu, pmtn|C_{max}$). This problem can be viewed as a generalization of two well (but mainly independently) studied scheduling problems, job shop with μ operations per job ($Jm|op \leq \mu|C_{max}$) and general set-constrained multiprocessor task scheduling ($Pm|set_j|C_{max}$). In classical job shop scheduling each operation is required to be processed on a single prespecified machine, so in terms of the

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 878–889, 2000.
© Springer-Verlag Berlin Heidelberg 2000

above formulation, for each operation there is only one processing mode corresponding to a single machine. In all of the different variants of multiprocessor task scheduling (dedicated, parallel, malleable, set-constrained), tasks are processed by subsets of processors, but there are no precedence relations specified for them. Therefore these can also be obtained (as special cases) from the previously defined general problem by requiring each task to consist of a single operation. Since both of the above special cases (jobshop and multiprocessor task scheduling) are NP-hard even if there are only a constant number of machines [6,7,9,16], it is natural to study how closely the optimum can be approximated by efficient algorithms. Focusing on the case where m and μ are fixed, we will show that there are linear time approximation schemes for the problem providing a unified extension of recent approximability results for both of the above special cases.

Job shop scheduling is considered to be one of the most difficult problems in combinatorial optimization, both from the theoretical and practical points of view. Even very restricted versions of the problem are strongly NP-hard [16]. For those instances where m and μ are fixed (the restricted case we are focusing on in this paper), Shmoys et al. [18] gave approximation algorithms that compute $(2+\epsilon)$-approximate solutions in polynomial time for any fixed $\epsilon > 0$. This result has recently been improved by Jansen et al. [14,15] who have shown that ϵ-approximate solutions of the problem can be computed in linear time for any fixed $\epsilon > 0$.

In classical scheduling theory, each task is processed by only one processor at a time. However recently, due to the rapid development of parallel computer systems, new theoretical approaches have emerged to model scheduling on parallel architectures. One of these is scheduling multiprocessor tasks, see e.g. [6,7]. The general (in contrast to dedicated, malleable, parallel) variant of non-preemptive scheduling for independent multiprocessor tasks on a fixed number of processors is denoted by $Pm|set_j|C_{max}$. Regarding the complexity, $P3|set_j|C_{max}$ is strongly NP-hard [3,9], thus already this restricted version has no fully polynomial approximation schemes, unless P=NP. For $Pm|set_j|C_{max}$, Bianco et al. [2] presented an approximation algorithm whose approximation ratio is bounded by m. Later Chen and Lee [4] improved their algorithm by achieving an approximation ratio $\frac{m}{2} + \epsilon$, for any $\epsilon > 0$. Until very recently, this was the best approximation result for the problem, and it was not known whether there is a polynomial-time approximation scheme or even a polynomial-time approximation algorithm with an absolute constant approximation guarantee. Chen and Miranda [5] have proposed a dynamic programming based polynomial-time approximation scheme for $Pm|set_j|C_{max}$ whose running time is $n^{O(1)}$ where the hidden constant in the big-O notation is proportional with $m^{O(m^2/\epsilon)}$. Independently from their work the authors have also proposed [11,13] a polynomial time approximation scheme for $Pm|set_j|C_{max}$ that computes an ϵ-approximate solution in $O(n)$ time for any fixed positive accuracy ϵ. The hidden constant in the previous bound - similarly as before - depends exponentially on the fixed parameters m and ϵ, but here the constant appears as a multiplicative factor of n and not as an exponent. Hence this linear programming based approach

leads to a substantially better (worst-case theoretical) running time than the one in [5], and also answers an open question of the latter paper by providing an polynomial time approximation scheme not based on dynamic programming.

In this paper we integrate many of the above mentioned recent results [5,11,13], [14,15] that have shown the existence of polynomial-time approximation schemes for various shop and multiprocessor scheduling problems. We present linear-time approximation schemes for both $Jm|set_{ij}, op \leq \mu|C_{max}$ and $Jm|set_{ij}, op \leq \mu, pmtn|C_{max}$ (under the assumption that m and μ are fixed), making these the currently known most general problems in shop and multiprocessor task scheduling with makespan minimization that posses polynomial-time approximation schemes. Some of the previous results [14,15] on job shop scheduling were based on a general vector summation result of Sevastianov. While this application of vector summation in scheduling is very interesting, it made the above papers highly dependent on the fairly involved underlying algorithm.

Sevastianov's algorithm computes a schedule for the job shop problem with makespan at most $L_{max} + \mu^3 m p_{max}$, where L_{max} is the maximum load of operations assigned to any machine and $p_{max} = \max_{j,i} p_{ji}$. The following simple example shows that we cannot use Sevastianov's algorithm in the general multiprocessor job shop model, in contrast to its applications in [10,14,15]. In these papers, it was applied only for operations with small p_{max} value to obtain schedules of length at most $(1+\epsilon)L_{max}$. Suppose that $m = 3$ and there are three types of tasks τ_1, τ_2, τ_3 each with $\mu = 1$ and one mode per operation. Also assume that each task of type τ_j requires processor set π_j, where $\pi_1 = \{1, 2\}$, $\pi_2 = \{2, 3\}$ and $\pi_3 = \{1, 3\}$. Consider instances where we have the same number n of tasks of each type and every operation is of length 3. Then the maximum load $L_{max} = 2n$ and the optimum makespan $OPT = 3n$. In this case, all operations are small and therefore no algorithm can achieve $(1 + \epsilon)L_{max}$.

Here we not only generalize and combine the previous results, but also show by introducing a new shifting procedure that the use of Sevastianov's algorithm can be avoided making the algorithms and the whole presentation simpler and selfcontained.

2 Non-preemptive Scheduling

In this section we consider the non-preemptive version of the problem, where a task - once started - has to be completed without interruption. Thus a schedule consists of a processor assignment $\rho_{ji} \in m_{ji}$ (i.e. one of the feasible processor sets) and a starting time τ_{ji} for each operation O_{ji} such that at any time there is no processor assigned to more than one operation, the operations $O_{j1}, \ldots, O_{j\mu}$ of task T_j are executed one after another, i.e. $\tau_{ji} + p_{ji}(\rho_{ji}) \leq \tau_{j,i+1}$ for every $1 \leq i < \mu$). The objective is to compute a non-preemptive schedule that minimizes the overall makespan $C_{max} = \max\{\tau_{j\mu} + p_{j\mu}(\rho_{j\mu}) : T_j \in \mathcal{T}\}$.

2.1 Snapshots, Relative Schedules, Blocks, and Configurations

Let k be a constant (depending on m, μ and ϵ) that will be specified later. First, the algorithm will select a subset $\mathcal{L}$ of k long tasks with the largest values $d_1 \geq d_2 \geq \ldots \geq d_k$ and $d_k \geq d_j$ for $k+1 \leq j \leq n$ where $d_j = \sum_{i=1}^{\mu} \min_{\rho \in m_{ji}} [p_{ji}(\rho)]$. Let $D = \sum_{j=1}^{n} d_j$, $\mathcal{S} = \mathcal{T} \setminus \mathcal{L}$, $M = \{1, \ldots, m\}$ and OPT the length of an optimum schedule. Then, we have $D/m \leq OPT \leq D$. By normalization (divide all execution times by D), we may assume without loss of generality that $D = 1$ and that $1/m \leq OPT \leq 1$.

A *processor assignment* for $\mathcal{L}$ is a mapping $f : \{O_{ji} | T_j \in \mathcal{L}\} \to 2^M$ with $f(O_{ji}) \in m_{ji}$. Two operations O_{ji} and $O_{j'i'}$ for $T_j, T_{j'} \in \mathcal{L}$ are *compatible* if $j \neq j'$ and if they are processed on different machines (i.e. $f(O_{ji}) \cap f(O_{j'i'}) = \emptyset$). For a given processor assignment of $\mathcal{L}$, a *snapshot* of $\mathcal{L}$ is a subset of compatible operations. A *relative schedule* $R = (f, M(1), \ldots, M(g))$ of $\mathcal{L}$ is a processor assignment f along with a sequence $M(1), \ldots, M(g)$ of snapshots of $\mathcal{L}$ such that: $M(1) = M(g) = \emptyset$; each operation O_{ji} of task $T_j \in \mathcal{L}$ occurs in a subsequence of consecutive snapshots $M(\alpha_{ji}), \ldots, M(\beta_{ji})$, $2 \leq \alpha_{ji} \leq \beta_{ji} < g$, (where $M(\alpha_{ji})$ is the first and $M(\beta_{ji})$ is the last snapshot that contains O_{ji}); for operations O_{ji} and $O_{j,i+1}$ of task $T_j \in \mathcal{L}$, $\beta_{ji} < \alpha_{j,i+1}$; and consecutive snapshots are different. We observe that g can be bounded by $2k\mu + 1$. For a snapshot $M(\ell)$, let $P(\ell) = \cup_{O_{ji} \in M(\ell), T_j \in \mathcal{L}} f(O_{ji})$ be the set of processors that are used by operations from long tasks during snapshot $M(\ell)$.

Each of these snapshots $M(\ell)$ will be divided inductively into a number of subintervals. For each $\ell = 1, \ldots, g$, let t_ℓ denote the length of snapshot $M(\ell)$, where we may assume that each $t_\ell \leq 1$. Let $\delta < 1$ be a constant that will be specified later. In the first step, we divide $M(\ell)$ into $1/\delta$ subintervals of length $\delta t_\ell \leq \delta$. We call these subintervals *blocks* of depth 1. Next, each block of depth 1 is divided into $\frac{1}{\delta}$ subintervals of length $\delta^2 t_\ell \leq \delta^2$. These subintervals are called blocks of depth 2. We iterate this subdivision μ times and end up for each snapshot $M(\ell)$ with $1/\delta^\mu$ blocks of depth μ (each with length $\delta^\mu t_\ell$). Let $M_\mu(\ell)$ denote the set of all blocks of depth μ in snapshot $M(\ell)$. The total number $\bar{g}_\mu$ of blocks of depth μ is bounded by $(2k\mu + 1)/\delta^\mu$.

For each block $q \in M_\mu(\ell)$, we consider a set of configurations: Let $P(q)$ be the processor set used by operations of long jobs in block q (i.e. $P(q) = P(\ell)$ for every block q in snapshot $M(\ell)$). Furthermore let $P_{q,i}$, $i = 1, \ldots, n_q$, denote the different partitions of $F(q) = M \setminus P(q)$, and let $\mathcal{P}_q = \{P_{q,1}, \ldots, P_{q,n_q}\}$. For each partition $P_{q,i}$ we introduce a variable x_{qi} to indicate the total length of $P_{q,i}$ where only processors of $F(q)$ are executing short tasks and each subset of processors $F \in P_{q,i}$ executes at most one operation of a short task in $\mathcal{S}$ at each time step in parallel.

Let $L_{q\rho}$ be the total processing time for all small operations in $\mathcal{S}$ executed on processor set ρ in block q. For each task $T_j \in \mathcal{S}$ we use a set of decision variables $y_{jab} \in [0, 1]$ for vectors $a \in A$ and $b \in B$, where $A = \{a : 1 \leq a_1 \leq a_2 \leq \ldots \leq a_\mu \leq \bar{g}_\mu\}$ and $B = \{b : b_i \in 2^M, b_i \neq \emptyset, i = 1, \ldots, \mu\}$. Variable $y_{jab} = 1$, if and only if each operation O_{jk} of task T_j is scheduled in block a_k on processor set b_k for each $1 \leq k \leq \mu$; otherwise y_{jab} represents the corresponding fraction of

task T_j's processing. For a given block q, processor set ρ, $a \in A$ and $b \in B$, let $K_{q\rho}^{ab} = \{k : 1 \le k \le \mu, a_k = q, b_k = \rho\}$. Then by using the previous notation, the load $L_{q\rho}$ can be written as: $\sum_{T_j \in \mathcal{S}} \sum_{a \in A, b \in B} \sum_{k \in K_{q\rho}^{ab}} p_{jk}(\rho)\ y_{jab}$.

2.2 Linear Programming Formulation

The linear program $LP(R)$ for a given relative schedule $R = (f, (M(1), \ldots, M(g))$ of $\mathcal{L}$ is as follows:

Minimize $\sum_{\ell=1}^{g} t_\ell$ **s.t.**

(1) $t_\ell \ge 0, \quad \ell = 1, \ldots, g,$
(2) $\sum_{\ell=\alpha_{ji}}^{\beta_{ji}} t_\ell = p_{ji}(f(O_{ji})), \quad \forall T_j \in \mathcal{L},\ i = 1, \ldots, \mu,$
(3) $y_{jab} \ge 0, \quad \forall T_j \in \mathcal{S},\ a \in A,\ b \in B,$
(4) $\sum_{a \in A, b \in B} y_{jab} = 1, \quad \forall T_j \in \mathcal{S},$
(5) $x_{qi} \ge 0, \quad \forall q \in M_\mu(\ell),\ i = 1, \ldots, n_q,$
(6) $\sum_{1 \le i \le n_q | \rho \in P_{q,i}} x_{qi} \ge L_{q\rho}, \quad \forall q \in M_\mu(\ell),\ \rho \in 2^M \setminus \{\emptyset\},$
(7) $\sum_{T_j \in \mathcal{S}} \sum_{a \in A, b \in B} \sum_{k \in K_{q\rho}^{ab}} y_{jab} p_{jk}(\rho) \le L_{q\rho}, \quad \forall q \in M_\mu(\ell), \rho \in 2^M \setminus \{\emptyset\},$
(8) $\sum_{i=1}^{n_q} x_{qi} = \delta^\mu t_\ell, \quad \forall q \in M(\ell),\ \ell = 1, \ldots, g,$
(9) $y_{jab} = 0, \quad \forall T_j \in \mathcal{S}$ with $b_k \notin m_{jk}$ or $b_k \cap P(a_k) \ne \emptyset$.

In (9) we set some of the variables y_{jab} to zero, since processors in $P(a_k)$ can not be used for operations of small tasks, and if $b_k \notin m_{jk}$, then it is a non-feasible processor set for operation O_{jk}.

Lemma 1. *The optimum of $LP(R)$ is not larger than the makespan of any schedule of $\mathcal{T}$ that respects the relative schedule R.*

Proof. Start with an optimum schedule S^* for $\mathcal{T}$ that respects R. First, we show how to define the variables y_{jab} for each small task $T_j \in \mathcal{S}$. Assume that S^* schedules each operation O_{jk} on processor set $\rho_{jk} \in m_{jk}$ (on a feasible processor set) for a time interval $p_{jk}(\rho_{jk})$ that spans consecutive blocks $q(s_{jk}), \ldots, q(e_{jk})$ where s_{jk} might be equal to e_{jk}. Let $f_{jk}(i)$ be the fraction of operation O_{jk} that is scheduled in block $q(i)$ and define $\bar{b}_k = \rho_{jk}$ for $1 \le k \le \mu$. Let $\bar{b} = (\bar{b}_1, \ldots, \bar{b}_\mu)$, $s_j = (s_{j1}, \ldots, s_{j\mu})$, and assign values to the variables $y_{ja\bar{b}}$ as follows: Set $y_{js_j\bar{b}} = r$, where $r = \min\{f_{jk}(s_{jk}) : 1 \le k \le \mu\}$. To cover the remaining $1 - r$ fraction of each operation, we assign values to the other $y_{ja\bar{b}}$ variables. To do this, we set $f_{jk}(s_{jk}) = f_{jk}(s_{jk}) - r$. For at least one operation O_{jk} the new value $f_{jk}(s_{jk})$ is zero; for those operations with $f_{jk}(s_{jk}) = 0$ we set $s_{jk} = s_{jk} + 1$. Then, we assign values to the new variable $y_{js_j\bar{b}}$ as above and repeat the procedure until $r = 0$. Each iteration of this procedure assigns a value to a different variable, since from one iteration to the next at least one block s_{jk} is changed. The assignment of values to variables y_{jab} generates the load $L_{q\rho}$ of small operations for each block q and subset ρ. Now consider a block q in snapshot $M(\ell)$ of length $\delta^\mu t_\ell$. The optimum schedule S^* gives us directly the partitions $P_{q,i}$ used inside block q and the corresponding lengths x_{qi}. The assignment of values to these variables satisfies the constraints of the linear program $LP(R)$ where the total length $\sum_{\ell=1}^{g} t_\ell$ is equal to the makespan of S^*. Therefore, the optimum objective value of $LP(R)$ is less than or equal to the makespan of S^*. □

Let OPT_R be the length of an optimum schedule that respects relative schedule R. Now we describe how to compute an approximate solution of $LP(R)$. First we guess the length s of an optimum schedule and add the constraint

(0) $\sum_{\ell=1}^{g} t_\ell \leq s,$

to $LP(R)$. Then, we replace (6) and (7) with the following constraints (one for each block q and subset ρ):

(6-7) $\sum_{T_j \in \mathcal{S}} \sum_{a \in A, b \in B} \sum_{k \in K_{q\rho}^{ab}} p_{kj}(\rho)\ y_{jab} - \sum_{1 \leq i \leq n_q | \rho \in P_{q,i}} x_{qi} + 1 \leq \lambda.$

The new linear program denoted by $LP(R, s, \lambda)$ has a special block angular structure. The blocks $B_j = \{y_j : y_{jab} \geq 0, \sum_{a,b} y_{jab} = 1\}$ for $T_j \in \mathcal{S}$ are simplicies (of fixed dimension), and $B_{|S|+1} = \{(t_\ell, x_{qi})|$ conditions $(0), (1), (2), (5), (8)\}$ is a block with only a constant number of variables and constraints. The coupling constraints are the inequalities (6-7). Note that for each q and ρ, the function $f_{q\rho} = \sum_{T_j \in \mathcal{S}} \sum_{a \in A, b \in B} \sum_{k \in K_{q\rho}^{ab}} p_{kj}(\rho)\ y_{jab} - \sum_{1 \leq i \leq n_q | \rho \in P_{q,i}} x_{qi} + 1$ is non-negative over the blocks, since $\sum_{1 \leq i \leq n_q | \rho \in P_{q,i}} x_{qi} \leq \delta^\mu t_\ell \leq \delta^\mu s \leq \delta^\mu \leq 1$.

The Logarithmic Potential Price Directive Decomposition Method developed by Grigoriadis and Khachiyan [8] can be used to get an $\bar{\epsilon}$ relaxed decision procedure for $LP(R, s, \lambda)$. This procedure either determines that $LP(R, s, 1)$ is infeasible, or computes a feasible solution of $LP(R, s, 1 + \bar{\epsilon})$. For any fixed m and $\bar{\epsilon} > 0$, the overall running time of the procedure is $O(n)$. We disregard the relative schedule R, if $LP(R, 1, 1)$ is infeasible. Using binary search on $s \in [1/m, 1]$ we can compute in a constant number $O(\log \frac{m}{\epsilon})$ of iterations a value $\bar{s} \leq OPT_R(1 + \frac{\epsilon}{8})$ such that $LP(R, \bar{s}, 1 + \bar{\epsilon})$ is feasible.

2.3 Rounding

In this subsection, we show how to generate a feasible schedule using an approximate solution of the previously defined linear program. For the solution obtained after the binary search on s, let $\bar{\epsilon}_{q\rho} = \sum_{T_j \in \mathcal{S}} \sum_{a \in A, b \in B} \sum_{k \in K_{q\rho}^{ab}} p_{kj}(\rho)\ y^*_{j,a,b} - \sum_{1 \leq i \leq n_q | \rho \in P_{q,i}} x^*_{q,i}$. The inequalities $(6-7)$ imply that for any block q and non-empty subset ρ, $\bar{\epsilon}_{q,\rho} \leq \bar{\epsilon}$. If $\bar{\epsilon}_{q,\rho} \leq 0$, then we have nothing to do for the pair (q, ρ). Let $\bar{L}_{q,\rho} = \sum_{1 \leq i \leq n_q | \rho \in P_{q,i}} x^*_{q,i}$ the free space for small tasks that use processor set ρ. In the first step, we shift a subset $\bar{S} \subseteq \mathcal{S}$ of small tasks to the end of the schedule such that $\sum_{T_j \in \mathcal{S} \setminus \bar{S}} \sum_{a \in A, b \in B} \sum_{k \in K_{q\rho}^{ab}} p_{kj}(\rho)\ y^*_{j,a,b} \leq \bar{L}_{q,\rho}$. Then, the subset $\mathcal{S} \setminus \bar{S}$ of remaining small tasks fits into the free space for the ρ-processor tasks. Notice that this step is not sufficient to generate a feasible schedule. In a second phase, we use a new technique to transform the schedule into a feasible one. In the following, we show how to compute $\bar{S}$ in linear time for any fixed m. First, we modify the y-components of the solution. The y-components of the solution of $LP(R, \bar{s}, 1 + \bar{\epsilon})$ can be considered as fractional assignments. The lengths of y are defined as $\tilde{L}_{q,\rho} = \sum_{T_j \in \mathcal{S}} \sum_{a \in A, b \in B} \sum_{k \in K_{q\rho}^{ab}} p_{kj}(\rho)\ y^*_{j,a,b}$, for each block q and subset ρ. For every q and ρ, we have $\tilde{L}_{q,\rho} \leq \bar{L}_{q,\rho} + \bar{\epsilon}$. Consider now the following system:

(a) $y_{jab} \geq 0, \quad \forall T_j \in \mathcal{S},\, a \in A,\, b \in B,$
(b) $\sum_{a\in A, b\in B} y_{jab} = 1, \quad \forall T_j \in \mathcal{S},$
(c) $\sum_{T_j\in\mathcal{S}} \sum_{a\in A,b\in B} \sum_{k\in K^{ab}_{q\rho}} p_{kj}(\rho)\, y_{jab} = \tilde{L}_{q,\rho}, \;\; \forall q \in M_\mu(\ell),\, \rho \in 2^M \setminus \{\emptyset\}.$

This system can be written also in matrix form $Cz = c, z \geq 0$. Using linear programming we can obtain a solution in polynomial time with only a constant number of fractional assignments. We give now a rounding technique that needs only linear time. Let us assume that the columns of C are indexed such that the columns corresponding to variables y_{jab} for each task T_j appear in adjacent positions. First, we may assume that variables y_{jab} with zero value and the corresponding columns are removed from z and C, respectively. We will show that there exists a constant size subset $\bar{C}$ of these columns in which the number of non-zero rows is smaller than the number of columns. The non-zero entries of $\bar{C}$ induce a singular matrix of constant size. Therefore, we can find a non-zero vector z' with $\bar{C}z' = 0$ in constant time. Let $\gamma > 0$ be the smallest value such that at least one component of the vector $z + \gamma z'$ is either zero or one (if necessary we augment z' by adding some zero entries).

We assume that each task T_j (during the procedure) has at least two columns in C. If task T_j has only one column in C, then the corresponding variable y_{jab} must have value 1. The number of inequalities of type (c) is bounded by the constant $K = (2^m - 1)(2k\mu + 1)/\delta^\mu$ [the first factor is the number of non-empty subsets and the second the number of blocks]. Let $\bar{C}$ be the set formed by the first $2K + 2$ columns of C. We note that at most $2K + 1$ rows of $\bar{C}$ have non-zero entries. By the above assumption on the number of columns for each job, at most $K + 1$ of these entries come from the constraints (b) and at most K non-zero entries come from the constraints (c). Let $\bar{C}'$ be the submatrix of $\bar{C}$ induced by the non-zero rows. Since $\bar{C}'$ has at most $2K + 1$ rows and exactly $2K + 2$ columns, the matrix $\bar{C}'$ is singular and there is a non-zero vector z' such that $\bar{C}'z' = 0$. Using linear algebra and the constant size of $\bar{C}'$, such a vector z' can be found in constant time. We can repeat this procedure until there are at most $2K + 1$ columns in C. Therefore, the total number of iterations is at most $|\mathcal{S}| \cdot K^\mu - 2K - 1$ and each iteration can be done in constant time. By the above argument, there are at most $2K + 1$ variables y_{jab} that may have values different from 0 or 1. Since at the end of the rounding procedure each task has either 0 or at least 2 columns in C, at most K tasks receive fractional assignments.

Lemma 2. *For the set $\mathcal{F}$ of small tasks with fractional assignments after the rounding procedure, it holds that $|\mathcal{F}| \leq K$.*

Let $\bar{a}^{(j)} \in A$ and $\bar{b}^{(j)} \in B$ the unique vectors such that $y_{j\bar{a}^{(j)}\bar{b}^{(j)}} = 1$ for $T_j \in \mathcal{S} \setminus \mathcal{F}$. In the next step, we compute for every non-empty subset $\rho \subseteq M$ and block q with $\bar{\epsilon}_{q\rho} > 0$ a subset $S_{q\rho} \subseteq \mathcal{S} \setminus \mathcal{F}$ of tasks such that the total execution length $\sum_{T_j\in S_{q\rho}} \sum_{1\leq k\leq\mu | \bar{a}^{(j)}_k = q, \bar{b}^{(j)}_k = \rho} p_{kj}(\rho) \geq \bar{\epsilon}_{q\rho}$, and there is one task $T_{j(q,\rho)} \in S_{q\rho}$ for which $\sum_{T_j\in S_{q\rho}\setminus\{T_{j(q,\rho)}\}} \sum_{1\leq k\leq\mu | \bar{a}^{(j)}_k = q, \bar{b}^{(j)}_k = \rho} p_{kj}(\rho) < \bar{\epsilon}_{q\rho}$. Let $\mathcal{U} = \{T_{j(q,\rho)} |$ block q, subset $\rho\}$. In total, we get a set $\mathcal{U} \cup \mathcal{F}$ of tasks with cardinality at most $2K$, and a subset $\mathcal{V} = (\bigcup_{q,\rho} S_{q\rho}) \setminus \mathcal{U}$ of total execution length

at most $\sum_{q,\rho} |\bar{\epsilon}_{q,\rho}| \leq \bar{\epsilon}K$. By choosing $\bar{\epsilon} = \frac{\epsilon}{4mK}$, the total execution length of $\mathcal{V}$ can be bounded by $\frac{\epsilon}{4m} \leq \frac{\epsilon}{4}OPT$. Using that $\bar{s} \leq (1+\epsilon/8)OPT_R$ this implies:

Lemma 3. *The objective function value of the computed linear program solution restricted to $\mathcal{T}' = \mathcal{T} \setminus (\mathcal{U} \cup \mathcal{F})$ is at most $OPT_R + \frac{3\epsilon}{8}OPT$, and $|\mathcal{U} \cup \mathcal{F}| \leq 2K$.*

In the next step, we have to show how to compute a feasible schedule for the tasks in $\mathcal{T}'' = \mathcal{T} \setminus (\mathcal{U} \cup \mathcal{V} \cup \mathcal{F})$. The other tasks in $(\mathcal{U} \cup \mathcal{V} \cup \mathcal{F})$ will be scheduled afterwards at the end of the schedule.

2.4 Shifting Procedure

The main difficulty is the case with more than one operations per task inside of a snapshot or a block. In this case, the algorithm in the last step might generate infeasibilities between these operations. Therefore, we introduce and use a shifting procedure for each snapshot to avoid this difficulty. Recall that we have a hierarchy of blocks of different depths (i.e. a block of depth i consists of $1/\delta$ blocks of depth $i+1$).

Consider one snapshot $M(\ell)$, $\ell \in \{1, \ldots, g\}$. Let $seq = (set_1, \ldots, set_\mu)$ be a fixed sequence of processor sets $set_i \subseteq M$, $set_i \neq \emptyset$ for $1 \leq i \leq \mu$. Furthermore, let $\mathcal{T}_{seq} \subseteq \mathcal{T}''$ be the set of small tasks T_j where operation O_{ji} is scheduled on processor set set_i for $1 \leq i \leq \mu$. For each task $T_j \in \mathcal{T}''$ such a sequence can be derived from the LP solution. Finally, let $\mathcal{T}_{seq}(\ell) \subseteq \mathcal{T}_{seq}$ be the subset of tasks T_j where at least one operation of the task is assigned to a block inside of snapshot $M(\ell)$, $1 \leq \ell \leq g$. Since the LP has generated for each $T_j \in \mathcal{T}_{seq}(\ell)$, a unique vector $(\bar{a}_1^{(j)}, \ldots, \bar{a}_\mu^{(j)})$ such that $\bar{a}_1^{(j)} \leq \ldots \leq \bar{a}_\mu^{(j)}$, for each $T_j \in \mathcal{T}_{seq}(\ell)$, a sequence of consecutive operations $O_{j,s_\ell^{(j)}}, \ldots, O_{j,e_\ell^{(j)}}$ lies inside snapshot $M(\ell)$. In the shifting procedure, we modify the assignments of blocks only for these operations.

For each snapshot $M(\ell)$, we insert different blocks of different depths. All inserted blocks will have the same set of free processors as $M(\ell)$. We add one block of depth 1 to the left side of the snapshot. The length of this block will be δt_ℓ. Next we add one block of depth 2 with length $\delta^2 t_\ell$ to the left side of each original block of depth 1. We repeat this process until depth $\mu - 1$. In iteration i, $1 \leq i \leq \mu - 1$, we have inserted $(\frac{1}{\delta})^{i-1}$ blocks of depth i. The total enlargement of the whole interval can be bounded by $\sum_{i=1}^{\mu-1} \frac{\delta^i t_\ell}{\delta^{i-1}} = \sum_{i=1}^{\mu-1} \delta t_\ell \leq \mu \delta t_\ell$. Next we apply our shifting procedure for the first $\mu - 1$ operations. The key idea is to shift the ith operation at least one block of depth i to the left but not into one of the blocks of depth $i-1$ to the left side. Notice that we do not consider inserted block of depth $1, \ldots, i-1, i+1, \ldots, \mu-1$ in iteration i. In the first iteration, we consider all 1st operations of tasks $T_j \in \mathcal{T}_{seq}(\ell)$ with $s_\ell^{(j)} = 1$. We start with the leftmost original block q of depth μ in $M(\ell)$ and shift all operations O_{j1} executed on set_1 into the first inserted block $q_{new}(1,1)$ to the left side. To do this, we reduce the length $\tilde{L}_{q,set_1}$ and store the free space $\sum_{T_j \in \mathcal{T}_{seq}(\ell), s_\ell^{(j)}=1} p_{j1}(set_1)$ in block q. The free space in block $q_{new}(1,1)$

is now $\delta t_\ell - \sum_{T_j \in \mathcal{T}_{seq}(\ell), s_\ell^{(j)}=1} p_{j1}(set_1)$. Then, we go over to the next original block to the right side and shift again the corresponding operations into block $q_{new}(1,1)$. We repeat this procedure until the block $q_{new}(1,1)$ is completely used corresponding set_1. At this time, we may interrupt one operation and shift the remaining part of this operation and the next operations into the first original block of depth μ with some free space. During the shifting we have to update the free spaces in the blocks. Notice that we do not use inserted blocks of depth $2, \ldots, \mu - 1$.

After the first iteration, every 1st operation of a task $T_j \in \mathcal{T}_{seq}(\ell)$ (with $s_\ell^{(j)} = 1$) is shifted at least one block of depth 1 to the left. On the other hand, several operation are preempted but the lengths $\tilde{L}_{q,set_1}$ are not increased. The total number of preemptions can be bounded by $1 + 1/\delta^\mu$. Now we go over to the second operations of tasks $T_j \in \mathcal{T}_{seq}(\ell)$ [only tasks with $s_\ell^{(j)} \leq 2 \leq e_\ell^{(j)}$ are considered]. Again, we start with the leftmost original block q of depth μ and shift all operations O_{j2} executed on set_2 into the first inserted block $q_{new}(2,1)$ of depth 2. Again, we reduce the length $\tilde{L}_{q,set_2}$ and store the free space generated by the processing times $p_{j2}(set_2)$ of all shifted operations O_{j2}. Again, we iteratively consider all blocks of depth μ inside the leftmost block of depth 1. Similar to above, the procedure shifts the second operations at least one block of depth 2 to the side. After considering the last block inside of the leftmost block of depth 1, we go over to the second block of depth 1. But now we shift the 2nd operations into the second inserted block $q_{new}(2,2)$ of depth 2. Notice that we do not shift operations into the neighbour block (to the left side) of depth 1. Using this idea, we eliminate infeasibilities between the first and second operations and we obtain different assigned blocks. Again, we have not increased the original values $\tilde{L}_{q,set_2}$ but some operations are preempted (during the procedure). The number of preemptions in the 2nd iteration is bounded by $1/\delta + 1/\delta^\mu$. After $\mu - 1$ iterations, we have $\sum_{i=1}^{\mu-1}(1/\delta)^{i-1} + \frac{1}{\delta^\mu} \leq \frac{2}{\delta^\mu}$ preemptions (using that $\mu \leq \frac{1}{\delta}$ and $\delta < 1$). Among all snapshots $M(\ell)$, this gives us at most $(2k\mu + 1)\frac{2}{\delta^\mu}$ preempted operations and an enlargement of the schedule of at most $\mu\delta OPT_R(1+\epsilon/8)$. We apply this shifting procedure for all job types. Since there are at most $(2^m - 1)^\mu$ different sequences of non-empty subsets of M, the number of preemptions is at most $(2k\mu + 1)\frac{2(2^m-1)^\mu}{\delta^\mu}$, and the total enlargement is bounded by $\mu\delta(2^m - 1)^\mu OPT_R(1 + \epsilon/8)$.

Let $\mathcal{X}$ be the set of small tasks that are preempted during the shifting procedure and let $\mathcal{T}''' = \mathcal{T}'' \setminus \mathcal{X}$. The tasks in $\mathcal{X}$ will be scheduled again at the end of the schedule. During the shifting procedure, we update the assignments of the blocks to the operations. Let $(\tilde{a}_1^{(j)}, \ldots, \tilde{a}_\mu^{(j)})$ be the generated new sequence of blocks. The main result of the shifting procedure is that $\tilde{a}_1^{(j)} < \ldots < \tilde{a}_\mu^{(j)}$. Remember that the number of blocks is increased by $\sum_{i=1}^{\mu-1}(\frac{1}{\delta})^{i-1} \leq \frac{1}{\delta^{\mu-1}}$ for each snapshot. On the other hand, operations (of tasks in $\mathcal{T}'''$) that are completely assigned to a new block can be executed one after another without preemptions and without generating conflicts [they have all the same processor set set_i in the new block]. This implies that in the final step we have to generate a schedule

(or pseudo schedule $PS(B)$) only for the original blocks of depth μ. If we define $\delta = \frac{\epsilon}{4\mu(2^m-1)^\mu}$, the total enlargement is bounded by $\frac{\epsilon}{4}(1+\frac{\epsilon}{8})OPT_R$.

2.5 Computation of Pseudo Schedules

In the final phase, the algorithm computes a pseudo schedule $PS(q)$ for each block q. We consider in this phase only the tasks in $\mathcal{T}'''$. The tasks in $\mathcal{T}'''$ and their operations are assigned unique to blocks. For each block q and non-empty subset $\rho \subseteq M$, we have a set of operations

$$\mathcal{O}_{q\rho} = \{O_{jk} | T_j \in \mathcal{T}''', \tilde{a}_k^{(j)} = q, \bar{b}_k^{(j)} = \rho\}.$$

The set $\mathcal{O}_{q\rho}$ will be scheduled in block q on subset ρ, where $\rho \subseteq F(q)$. Let $\hat{L}_{q\rho} = \sum_{T_j \in \mathcal{T}'''} \sum_{1 \le k \le \mu, O_{jk} \in \mathcal{O}_{q\rho}} p_{jk}(\rho)$, the total processing time of operations in $\mathcal{O}_{q\rho}$. Using the steps before, we know that $\hat{L}_{q\rho} \le \bar{L}_{q\rho} = \sum_{1 \le i \le n_q, \rho \in P_{q,i}} x^*_{q,i} \le \delta^\mu \cdot t_\ell$. From the left to the right, we place now the operations of $\mathcal{O}_{q\rho}$ on the free processors in $F(B)$. To do this, we consider each partition $P_{q,i}$ with value $x^*_{q,i} > 0$. For each set ρ in the partition $P_{q,i}$, we place a sequence of operations that use processor set ρ with total execution length $x^*_{q,i}$. If necessary, the last (and first) operation assigned to ρ is preempted. Since $\sum_{1 \le i \le n_q, \rho \in P_{q,i}} x^*_{q,i} \ge \hat{L}_{q\rho}$, the procedure completely schedules all operations in $\mathcal{O}_{q\rho}$ for every $\rho \subseteq F(q)$, and it runs in $O(n)$ time. Let $\mathcal{W}_q$ be the set of preempted (and therefore incorrectly scheduled) tasks in $PS(q)$, and let $\mathcal{W} = \cup_q \mathcal{W}_q$. The following lemma gives an upper bound on the cardinality of $\mathcal{W}$.

Lemma 4. $|\mathcal{W}| \le m^{m+1}(2k\mu+1)/\delta^\mu$.

We remove the tasks in $\mathcal{W}$ of tasks and execute them at the end of the schedule. This gives a feasible schedule of length at most $OPT_R(1+\frac{3\epsilon}{4}) + \mathcal{S}(\mathcal{U} \cup \mathcal{F} \cup \mathcal{W} \cup \mathcal{X})$, where $\mathcal{S}(\mathcal{Y})$ denotes the length of subset $\mathcal{Y} \subseteq \mathcal{T}$, where each task $T_j \in \mathcal{Y}$ is executed on the fastest processor set corresponding to the d_j value. In the following, we will bound the length of $\mathcal{U} \cup \mathcal{F} \cup \mathcal{W} \cup \mathcal{X}$. The previous bounds also imply the following inequality:

Corollary 1. $|\mathcal{U} \cup \mathcal{F} \cup \mathcal{W} \cup \mathcal{X}| \le [15m^{m+1}(4\mu 2^{m\mu})^{\mu+1}/\epsilon^\mu] \cdot k$.

Finally, we have to choose k such that the removed tasks can be scheduled with length at most $\frac{\epsilon}{4m} \le \frac{\epsilon}{4}OPT$. This can be achieved by a combinatorial lemma proved by the authors in [12]. This concludes the proof of our main result:

Theorem 1. *For any fixed m and μ, there is an approximation scheme for $Jm|set_{ij}, op \le \mu|C_{max}$ (the general multiprocessor job shop scheduling problem) that for any fixed $\epsilon > 0$ computes an ϵ-approximate schedule in $O(n)$ time.*

Recent results showing the existence of (linear-time) approximation schemes for $Jm|op \le \mu|C_{max}$ (job shop problem) in [14,15], $Pm|fix_j|C_{max}$ (dedicated machines) in [1], $Pm|size_j|C_{max}$ (parallel scheduling) in [11], $Pm|fctn_j|C_{max}$ (malleable tasks) in [11], $Pm|set_j|C_{max}$ in [5,13] and $Rm||C_{max}$ (unrelated parallel machines) in [12] can all be obtained as special cases of Theorem 1.

3 Preemptive Scheduling

In the *preemptive* model of scheduling, the processing of any job is allowed to be interrupted any time at no cost and restarted later. Two variants of this model can be considered according to the possibility of changing processor sets after preemptions. Preemptive scheduling *without migration* requires each operations to be processed by the same processor set during the whole schedule, i.e. if the operation is preempted, later it has to be resumed on the same set of processors. Preemptive scheduling *with migration* allows preempted operations to be later resumed on a different subset of processors than the one(s) used before.

In both variants we use fixed sequences of start- and end-events for operations of long jobs (to allow preemptions). For the model without migration, we fix in the relative schedule the assignment of processor sets for all operations of long jobs, but we use the same type of variables y_{jab} for both long and small jobs. For schedules with possible migration, we also have to allow the change of processor sets for operations of long jobs. To do this, we split each snapshot into 2^m intervals corresponding to the different subsets of M such that during interval I_p only processors in the corresponding processor set P_p can be used for long operations, and all the other processors in $M \setminus P_p$ are used for small operations. Furthermore, we partition each of these intervals into different subintervals corresponding to the different mappings f of processors to operations that map the processors in P_p to operations of long jobs that are feasible for snapshot ℓ. By using slightly more complicated linear programming formulations and then applying the shifting procedure, the following results can be shown. (The proof along with the complete descriptions of the linear programming formulations will be given in the full version of this paper.)

Theorem 2. *For any fixed m and μ, there is a polynomial-time approximation scheme for $Jm|set_{ij}, op \leq \mu, pmtn|C_{max}$ (both with and without migration) that for any fixed $\epsilon > 0$ computes an ϵ-approximate schedule in $O(n)$ time.*

4 Conclusion

In this paper we have proposed linear-time approximation schemes for preemptive and non-preemptive variants of the general multiprocessor job shop scheduling problem, showing that this is the most general currently known problem in shop and multiprocessor task scheduling with makespan minimization that posses polynomial time approximation schemes. This result also provides integrations (extensions) of many recent results in both job shop [14,15] and multiprocessor task scheduling [5,11,13]. We have introduced a general shifting procedure which is an essential component of the algorithms above and will likely to find further applications in scheduling. This new procedure also allowed us to avoid using the quite involved vector summation algorithm of Sevastianov that is a key ingredient in the previously proposed approximation schemes for job shop scheduling [14,15], making the algorithms and presentation simpler and self-contained.

References

1. A.K. Amoura, E. Bampis, C. Kenyon and Y. Manoussakis, Scheduling independent multiprocessor tasks, *Proceedings of the 5th European Symposium on Algorithms* (1997), LNCS 1284, 1-12.
2. L. Bianco, J. Blazewicz, P. Dell Olmo and M. Drozdowski, Scheduling multiprocessor tasks on a dynamic configuration of dedicated processors, *Annals of Operations Research* 58 (1995), 493-517.
3. J. Blazewicz, P. Dell Olmo, M. Drozdowski and M. Speranza, Scheduling multiprocessor tasks on the three dedicated processors, *Information Processing Letters* 41 (1992), 275-280.
4. J. Chen and C.-Y. Lee, General multiprocessor tasks scheduling, *Naval Research Logistics*, in press.
5. J. Chen and A. Miranda, A polynomial time approximation scheme for general multiprocessor job scheduling, *Proceedings of the 31st ACM Symposium on the Theory of Computing* (1999), 418-427.
6. M. Drozdowski, Scheduling multiprocessor tasks - an overview, *European Journal on Operations Research*, 94 (1996), 215-230.
7. J. Du and J. Leung, Complexity of scheduling parallel task systems, *SIAM Journal on Discrete Mathematics*, 2 (1989), 473-487.
8. M.D. Grigoriadis and L.G. Khachiyan, Coordination complexity of parallel price-directive decomposition, *Mathematics of Operations Research* 21 (1996), 321-340.
9. J.A. Hoogeveen, S.L. van de Velde and B. Veltman, Complexity of scheduling multiprocessor tasks with prespecified processor allocations, *Discrete Applied Mathematics* 55 (1994), 259-272.
10. K. Jansen, M. Mastrolilli and R. Solis-Oba, Approximation algorithms for flexible job shop problems, *Proceedings of the 4th Latin American Theoretical Informatics* (2000), LNCS 1776, Springer Verlag, 68-77.
11. K. Jansen and L. Porkolab, Linear-time approximation schemes for scheduling malleable parallel tasks, *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms* (1999), 490-498.
12. K. Jansen and L. Porkolab, Improved approximation schemes for scheduling unrelated parallel machines, *Proceedings of the 31st ACM Symposium on the Theory of Computing* (1999), 408-417.
13. K. Jansen and L. Porkolab, General multiprocessor task scheduling: approximate solution in linear time, *Proceedings of the 6th International Workshop on Algorithms and Data Structures* (1999), LNCS 1663, Springer Verlag, 110-121.
14. K. Jansen, R. Solis-Oba and M.I. Sviridenko, Makespan minimization in job shops: a polynomial time approximation scheme, *Proceedings of the 31st ACM Symposium on the Theory of Computing* (1999), 394-399.
15. K. Jansen, R. Solis-Oba and M.I. Sviridenko, A linear time approximation scheme for the job shop scheduling problem, *Proceedings of the 2nd Workshop on Approximation Algorithms* (1999), LNCS 1671, Springer Verlag, 177-188.
16. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan and D.B. Shmoys, Sequencing and scheduling: algorithms and complexity, in: Handbook of Operations Research and Management Science, Vol. 4, North-Holland, 1993, 445-522.
17. S.A. Plotkin, D.B. Shmoys and E. Tardos, Fast approximation algorithms for fractional packing and covering problems, *Mathematics of Operations Research* 20 (1995), 257-301.
18. D.B. Shmoys, C. Stein and J. Wein, Improved approximation algorithms for shop scheduling problems, *SIAM Journal on Computing*, 23 (1994), 617-632.

The Many Faces of a Translation

Pierre McKenzie[1⋆], Thomas Schwentick[2],
Denis Thérien[3⋆], and Heribert Vollmer[4]

[1] Informatique et recherche opérationnelle, Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada
[2] Institut für Informatik, Johannes-Gutenberg-Universität Mainz, 55099 Mainz, Germany
[3] School of Computer Science, McGill University, 3480 University Street, Montréal (Québec), H3A 2A7 Canada
[4] Theoretische Informatik, Universität Würzburg, Am Hubland, 97074 Würzburg, Germany

Abstract. First-order translations have recently been characterized as the maps computed by aperiodic single-valued nondeterministic finite transducers (NFTs). It is shown here that this characterization lifts to "**V**-translations" and "**V**-single-valued-NFTs", where **V** is an arbitrary monoid pseudovariety. More strikingly, 2-way **V**-machines are introduced, and the following three models are shown exactly equivalent to Eilenberg's classical notion of a bimachine when **V** is a group variety or when **V** is the variety of aperiodic monoids: **V**-translations, **V**-single-valued-NFTs and 2-way **V**-transducers.

1 Introduction

The regular languages have been characterized in many ways, in particular using DFAs, NFAs, and 2-way DFAs [RS59,She59], monadic second order logic [Büc60,Tra61] and finite semigroups [Eil76b], with an incisive algebraic parametrization arising from the latter. For instance, aperiodic DFAs recognize exactly the star-free languages [Sch65], and these are precisely captured by FO, i.e. first-order logic with order [MP71].

In a separate vein, renewed interest in FO was sparked by the development of descriptive complexity (see [Imm99]), and the old concept of a translation became the natural way to reduce one problem to another. FO-translations and projections, in particular, became the lowest level reductions used in the theory.

Experience with the regular languages would suggest characterizing FO-translations using aperiodic deterministic finite transducers. But this fails, because such transducers cannot even map $w_1 w_2 \cdots w_n$ to w_n^n. Lautemann and three of the present authors [LMSV99] showed that the appropriate automata-theoretic model required to capture FO-translations is the aperiodic *nondeterministic* transducer, restricted to output a unique string on any input.

⋆ Supported by NSERC of Canada and by FCAR du Québec.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 890–901, 2000.
© Springer-Verlag Berlin Heidelberg 2000

Here we extend this result in several directions. We first generalize from FO-translations to **V**-translations, where **V** is an arbitrary monoid variety (a monoid variety is the most convincing notion imaginable of a natural set of monoids). FO-translations are obtained when **V** is the variety of aperiodic monoids, and other special cases of such **V**-logical formulas were studied before, for example when $\exists$ and $\forall$ quantifiers were replaced by MOD_q quantifiers in the work of [STT88,BCST92,Str94].

Second, we define an NFA to be a **V**-NFA if applying the subset construction to it yields (the familiar notion of) a **V**-DFA.

Third, we consider 2-way DFAs, and define what constitutes a 2-way **V**-machine. This is delicate because the appropriate definition of the extended transition function of such an automaton is not obvious, which in fact explains the absence of an algebraic treatment of 2-way DFAs in the literature.

Our main result is a striking equivalence between the above notions and the old notion of a bimachine developed by Eilenberg [Eil76a, p. 320]:

Theorem 1. *Let $f\colon \Sigma^* \to \Gamma^*$, let* **V** *be the variety* **A** *of aperiodic monoids or an arbitrary variety of groups. The following statements are equivalent:*

(i) f is a **V***-translation.*
(ii) There is a sequential **V***-bimachine that computes f.*
(iii) There is a single-valued nondeterministic **V***-transducer that computes f.*
(iv) There is a 2-way **V***-machine that computes f.*

Moreover, for the much larger class of varieties **V** *closed under reversal, $(i) \Leftrightarrow (ii) \Leftrightarrow (iii)$.*

In the case **VA**, Theorem 1 states that FO-translations, aperiodic bimachines, single-valued nondeterministic aperiodic transducers, and deterministic aperiodic 2-way transducers, all compute exactly the same functions.

Intuitively, the main ingredients of Theorem 1 are that NFAs can simulate bimachines by guessing the behavior of the second machine on the unread part v of an input string uv (and they can simulate FO by guessing the possible sets of inequivalent formulas consistent with u satisfied by uv). The link with 2-way DFAs, in the non-group case, comes from simulating NFAs in the slick way developed by Hopcroft and Ullman [HU67], yet a considerable amount of care is needed to ensure that the simulation preserves the required algebraic properties. (In this paper, we limit our claim to the statement of Theorem 1, although we suspect that the equivalence between the four models of the theorem holds for many more varieties.)

Section 2 contains preliminaries and the precise definitions of our computation models. Section 3, presenting our results, first addresses the models equivalent for any **V** closed under reversal, and then we bring in the 2-way model. Due to space restrictions, most proofs have to remain sketchy or are even omitted from this abstract. For full proofs we refer the reader to [MSTV00].

2 Definition of Models and Preliminary Results

An associative binary operation on a set containing an identity for this operation defines a *monoid*. By a monoid *variety*, we mean a pseudovariety in the sense of Straubing [Str94, pp. 72ff] or Eilenberg [Eil76b, pp. 109ff]: it is a set of monoids closed under the finite direct product, the taking of submonoids and of homomorphic images. Examples of varieties include the commutative monoids, the groups $\mathbf{G}$, the aperiodics $\mathbf{A}$ (i.e., the monoids containing only trivial groups), the solvable groups $\mathbf{G}_{sol}$, or any set of monoids satisfying a set of identities, as we explain next.

Let $U\{u_1, u_2, u_3, \dots\}$ be a countable alphabet of *variables*, and let $l, r \in U^*$. Then an equation of the form lr is a *monoid identity*. A monoid M satisfies the above identity if, for every homomorphism $h\colon U^* \to M$, we have $h(l)h(r)$. Monoid identities are used to define pseudovarieties of monoids, we refer the reader to the excellent presentation in [Str94, Chapter V.6]. Precisely, a set $\mathbf{V}$ of monoids is a variety, if and only if there exists a sequence $(l_i r_i)_i \geq 1$ of equations such that a monoid M belongs to $\mathbf{V}$ iff it satisfies all but finitely many equations $l_i r_i$, see [Eil76b, Sect. V.2].

If M is a finite automaton (one-way or two-way) such that the transformation monoid of M satisfies the identity lr, then we say that lr holds in M.

Deterministic finite automata are, as usual [HU79, p. 17], given by $M(S, \Sigma, \delta, s, F)$, where the different components denote the state set, the input alphabet, the transition function, the initial state, and the set of final states, respectively. The extended transition function [HU79, p. 17] of M is denoted by $\hat{\delta}$. The *transformation monoid* of M is the set $\left\{ \hat{\delta}(\cdot, w)\colon S \to S \;\middle|\; w \in \Sigma^* \right\}$ with the operation of composition of functions. We say that M is a $\mathbf{V}$-DFA if its transformation monoid is in $\mathbf{V}$. A $\mathbf{V}$-language is a language accepted by a $\mathbf{V}$-DFA.

An equivalence relation $\sim$ on Σ^* is a *congruence* if $u \sim v$ implies $(\forall x, y \in \Sigma^*)[xuy \sim xvy]$. A congruence $\sim$ induces a monoid $\Sigma^*/\!\sim$ isomorphic to the transformation monoid of the pre-automaton $(\Sigma^*/\!\sim, \Sigma, ([u]_\sim, a) \mapsto [ua]_\sim)$. An example of a congruence is the syntactic congruence of a language L: $u \sim_L v$ iff $(\forall x, y \in \Sigma^*)[xuy \in L \text{ iff } xvy \in L]$. Let a DFA $M(S, \Sigma, \delta, s, F)$ be given. Another example of a congruence is: $u \sim_M v$ iff $\hat{\delta}(\cdot, u)\hat{\delta}(\cdot, v)$. Then $\Sigma^*/\!\sim_M$ is isomorphic to the transformation monoid of M. Writing $\sim_{pq}$ for the syntactic congruence of the language $L(p, q)\left\{w \;\middle|\; \hat{\delta}(p, w)q\right\}$, there is an injective morphism $\Sigma^*/\!\sim_M \longrightarrow \Pi_{p,q \in S}(\Sigma^*/\!\sim_{pq})$ and a surjective morphism $\Sigma^*/\!\sim_M \longrightarrow \Sigma^*/\!\sim_{pq}$. These facts can be shown to imply that M is a $\mathbf{V}$-DFA iff $L(p, q)$ is a $\mathbf{V}$-language for each $p, q \in S$.

2.1 FO-Translations and V-Translations

We consider first-order logic with linear order, denoted by $\mathbf{FO}$ (in the literature, this is often denoted $\mathbf{FO}[<]$, see, e.g., [Imm99]). We restrict our attention to *string signatures*, i. e., signatures of the form $\langle C_{a_1}, \dots, C_{a_s} \rangle$, where all the predicates C_{a_i} are unary, and in every structure $\mathcal{A}$, $\mathcal{A} \models C_{a_i}(j)$ iff the jth symbol

in the input is the letter a_i. Such structures are thus words over the alphabet $\Sigma\{a_1, \ldots, a_s\}$, and first-order variables range over positions within such a word, i. e. from 1 to the word length n. A formula from this logic is called a formula over Σ. Our basic formulas are built from variables in the usual way, using the Boolean connectives $\{\wedge, \vee, \neg\}$, the relevant predicates C_{a_i} together with $\{, <\}$, the quantifiers $\{\exists, \forall\}$, and parentheses.

Let Σ be as above, and consider a second alphabet $\Gamma\{b_1, \ldots, b_{t+1}\}$. Fix an order $b_1 < \cdots < b_{t+1}$ on Γ. Let $\varphi_1, \ldots, \varphi_t$ be first-order formulas over Σ, each with one free variable x. These formulas define a mapping $[\varphi_1, \ldots, \varphi_t]\colon \Sigma^* \to \Gamma^*$ as follows: Let $ww_1 \cdots w_n \in \Sigma^n$. Then, $[\varphi_1, \ldots, \varphi_t](w)v_1 \cdots v_n \in \Gamma^n$, where

$$v_i \begin{cases} b_1 & \text{if } w \models \varphi_1(i), \\ b_2 & \text{if } w \models \neg\varphi_1(i) \wedge \varphi_2(i), \\ & \ldots \\ b_{t+1} & \text{if } w \models \neg\varphi_1(i) \wedge \neg\varphi_2(i) \wedge \cdots \wedge \neg\varphi_t(i), \end{cases}$$

for $1 \leq i \leq n$. In this definition, $\varphi_j(i)$ is the formula that is obtained when in φ_j variable x takes value i.

This function is called a **FO**-translation or first-order definable translation, see [LMSV99]. (In the more general case, where the formulas $\varphi_1, \ldots, \varphi_t$ are allowed to have more than one free variable, these functions are called first-order reductions or first-order interpretations, see [Imm99].)

Now let Σ and Γ be as above and $A \subseteq \Gamma^*$. Then we say that $Q_A x[\varphi_1, \ldots, \varphi_t]$ is a $Q_A^{\mathrm{un}}\mathbf{FO}$ formula (over Σ). For $w \in \Sigma^*$, we define $w \models Q_A x[\varphi_1, \ldots, \varphi_t]$ if $[\varphi_1, \ldots, \varphi_t](w) \in A$.

In this paper, the sets A defining quantifiers will most of the time be *monoid word problems*. For a monoid M and a subset $F \subseteq M$, we define the word problem $W(M, F)$ as the language $\{\, m_1 \cdots m_k \in M^* \mid k \in \mathbb{N}, m_1 \circ \cdots \circ m_k \in F \,\}$, where $\circ$ denotes multiplication in M.

Let $\mathbf{V}$ be a pseudovariety of monoids. We will now use $\mathbf{V}$ word problems to define translations as follows. First, define $\mathbf{V}$-formulas inductively by:

- Every quantifier-free formula is a $\mathbf{V}$-formula.
- If φ is quantifier-free, x a variable, $M \in \mathbf{V}$, and $F \subseteq M$, then $Q_{W(M,F)} x\ \varphi$ is a $\mathbf{V}$-formula.
- Every Boolean combination of $\mathbf{V}$-formulas is a $\mathbf{V}$-formula.

A $\mathbf{V}$-*translation* is a translation $[\varphi_1, \ldots, \varphi_t]$, where $\varphi_1, \ldots, \varphi_t$ are $\mathbf{V}$-formulas with one free variable.

Examples. The quantifier $\exists$ is the quantifier $Q_{W(OR,\{1\})}$, where OR is the aperiodic monoid defined by the binary OR. Straubing [Str94] surveys an elegant theory in which FO is supplemented with "monoidal" quantifiers such as MOD_q, where $\mathrm{MOD}_q x\, \varphi$ holds iff $\varphi(x)$ holds in a multiple of q word positions x. Such quantifiers are $Q_{W(Z_q,\{0\})}$ quantifiers. Hence $[Q_{W(Z_q,\{0\})} y(y < x \wedge C_a y)]$ is a translation, mapping $w_1 w_2 \cdots w_n \in \{a, b, c\}^*$ to $v_1 \cdots v_n \in \{0, 1\}^n$, such that $v_i 1$ iff $w_1 \cdots w_{i-1}$ contains a multiple of q occurrences of a. Other monoidal quantifiers, e.g., $Q_{W(S_5,\{e\})}$ where S_5 is the (nonsolvable) symmetric group on

5 points (see, e.g., [BIS90]), have been used in the literature to capture the complexity class NC^1 using FO.

2.2 Bimachines and Bisequential Functions

Eilenberg [Eil76a, p. 320] defined a *bimachine* to be a triple $M(M_1, M_2, g)$ where $M_1(S_1, \Sigma, \delta_1, s_1)$, $M_2(S_2, \Sigma, \delta_2, s_2)$ are DFAs without final states, and $g\colon S_1 \times S_2 \times \Sigma \to \Gamma$. For $i \in \{1, \ldots, |w|\}$, the output of M on input $ww_1 \cdots w_n$ at position i is $o_M(w, i) g(\hat{\delta}_1(s_1, w_1 \cdots w_{i-1}), \hat{\delta}_2(s_2, w_n \cdots w_{i+1}), w_i)$. The *output* of M on input w is $T_M(w) o_M(w, 1) \cdots o_M(w, n)$. M is a $\mathbf{V}$-bimachine if the transformation monoids of both M_1 and M_2 are in $\mathbf{V}$.

We define a *bisequential function* as a pair $f(\alpha, m)$, where α is a congruence over some alphabet Σ, and m is a function $m\colon \Sigma^*/\alpha \times \Sigma^*/\alpha \times \Sigma \to \Gamma$ for some alphabet Γ, and for all $ww_1 \cdots w_n \in \Sigma^*$, $f(w) v_1 \cdots v_n$ where $v_i m\big([w_1 \cdots w_{i-1}]_\alpha, [w_{i+1} \cdots w_n]_\alpha, w_i\big)$. We say that f is a $\mathbf{V}$-bisequential function if the quotient set Σ^*/α with the usual multiplication is in $\mathbf{V}$.

Observe that if $\mathbf{V}$ is closed under reversal, a function is $\mathbf{V}$-bisequential if and only if it is computed by some $\mathbf{V}$-bimachine. Indeed, to compute the bisequential function $f(\alpha, m)$, define $M(M_1, M_2, m)$, where $M_1(S_1, \Sigma, \delta_1, s_1)$ and $M_2(S_2, \Sigma, \delta_2, s_2)$ are given by $S_1 S_2 \Sigma^*/\alpha$, $s_1 s_2 [\varepsilon]_\alpha$, $\delta_1\big([w]_\alpha, a\big)[wa]_\alpha$ and $\delta_2\big([w]_\alpha, a\big)[aw]_\alpha$ for all $a \in \Sigma$, $w \in \Sigma^*$. The reader may check that fT_M and that the transformation monoids of M_1 and M_2 are in $\mathbf{V}$.

The converse, given M to find a $\mathbf{V}$-bisequential function f such that $T_M f$, is proved in a very similar way.

2.3 Single-Valued Nondeterministic Transducers

In [LMSV99], a *nondeterministic finite transducer* is defined to be a tuple $M(S, \Sigma, \Gamma, \delta, I, F)$, where S is the set Q of *states*, Σ is the *input alphabet*, Γ is the *output alphabet*, $\delta \subseteq Q \times \Sigma \times \Gamma \times Q$ is the *transition relation*, $I \subseteq Q$ is the set of *initial states* and $F \subseteq Q$ is the set of *final states*. For a string $ww_1 \cdots w_n \in \Sigma^*$ we define the *set* $T_M(w)$ *of outputs of* M *on input* w as follows. A string $v \in \Gamma^*$ of length n is in $T_M(w)$, if there is a sequence $s_0, s_1, \ldots, s_n$ of states, such that $s_0 \in I$, $s_n \in F$ and, for every i, $1 \le i \le n$, we have $(s_{i-1}, w_i, v_i, s_i) \in \delta$.

We say that a nondeterministic finite transducer $(S, \Sigma, \Gamma, \delta, I, F)$ is a $\mathbf{V}$-*transducer*, or that a NFA $(S, \Sigma, \delta, s_0, F)$ is a $\mathbf{V}$-*NFA*, if applying the subset construction to the pre-NFA (S, Σ, δ) yields a pre-DFA $(2^S, \Sigma, \delta')$ whose transformation monoid is in $\mathbf{V}$.

Proposition 2. *The following statements are equivalent for a transducer M having (S, Σ, δ) as pre-NFA:*
(i) M is a $\mathbf{V}$-transducer,
(ii) For each $p, q \in S$, the language $L(p, q)$ accepted by the NFA $(S, \Sigma, \delta, p, \{q\})$ is a $\mathbf{V}$-language.

We say that a nondeterministic transducer M is *single-valued*, if, for every $ww_1 \cdots w_n \in \Sigma^*$ there is a single computation $(s_0, w_1, v_1, s_1)(s_1, w_2, v_2, s_2) \cdots$

(s_{n-1}, w_n, v_n, s_n) such that $s_0 \in I$, $s_n \in F$, and $(s_{i-1}, w_i, v_i, q_i) \in \delta$ for $1 \leq i \leq n$. Note that in this case, $|T_M(w)|1$ for all $w \in \Sigma^*$; if $T_M(w)\{u\}$, then we write $T_M(w)u$.

Remark 3. We remark that the definition above of single-valuedness is different from the one given in [LMSV99], since there it was only required that the set of values $T_M(w)$ consists of at most one word, for every $w \in \Sigma^*$. This still leaves the possibility that there may be different paths in M for a particular input w which produce the same output, which is forbidden in the present definition. [LMSV99] equate the power of aperiodic machines according to their definition with that of **FO**-translations. Since our main result in Sect. 3 gives an analogous statement for aperiodic machines according to our stricter definition, we conclude that for the aperiodic case, both definitions coincide.

2.4 Two-Way Automata

A two-way automaton with output is a 7-tuple $M(\mathcal{L} \uplus \mathcal{R}, \Sigma, \Gamma, \delta, \lambda, l_0, F)$, where

- the set S of states is the disjoint union $\mathcal{L} \uplus \mathcal{R}$ of a set $\mathcal{L}$ (the states "entered from the left") and a set $\mathcal{R}$ (the states "entered from the right"),
- $l_0 \in \mathcal{L}$ is the *initial state*,
- $F \subseteq S$ is the set of *final states* which, as in the case of one-way transducers, will play no role in defining the operation of a two-way automaton with output,
- Σ is the *input alphabet*, Γ is the *output alphabet*,
- $\delta\colon (S \times \Sigma) \cup (\mathcal{L} \times \{\lhd\}) \cup (\mathcal{R} \times \{\rhd\}) \to S$ is a total *transition function*, where $\rhd \notin \Sigma$ and $\lhd \notin \Sigma$ are the *leftmarker* and the *rightmarker* respectively,
- $\lambda\colon (S \times \Sigma) \to \Gamma$ is the *output function*.

The meaning of $\delta(s, \sigma) \in \mathcal{L}$ is that M in state s scanning σ moves its head to the *right* upon entering state $\delta(s, \sigma)$; M moves its head to the *left* when $\delta(s, \sigma) \in \mathcal{R}$.

The *initial configuration* of M on input $ww_1w_2 \cdots w_n \in \Sigma^*$ is the situation in which the state of M is l_0 and M scans w_1 within the string $\rhd w_1 w_2 \cdots w_n \lhd$ (M scans $\lhd$ when $|w|0$). We say that M eventually exits $\rhd w \lhd$ if it eventually encounters a transition $\delta(r, \rhd) \in \mathcal{R}$ or a transition $\delta(l, \lhd) \in \mathcal{L}$ (of course M will generally bounce off the end markers several times before exiting). We require that, for any $ww_1w_2 \cdots w_n \in \Sigma^*$,

- M from its initial configuration on w eventually leaves every w_i to the right, $1 \leq i \leq n$, and eventually exits $\rhd w \lhd$; this is analogous to the (unspoken) requirement that a one-way transducer must traverse its input and halt,
- from *any* state $l \in \mathcal{L}$ scanning w_1, M eventually exits $\rhd w \lhd$; this requirement is analogous to the (unspoken) fact that a one-way transducer eventually runs out of input regardless of its initial configuration,
- from *any* state $r \in \mathcal{R}$ scanning w_n, M also eventually exits $\rhd w \lhd$; this is justified by the natural desire to maintain symmetry between left and right in a two-way machine.

Each $ww_1w_2\cdots w_n \in \Sigma^*$ coerces M into a behavior described by a *behavior function*

$$\delta_w : \mathcal{L} \uplus \mathcal{R} \to \mathcal{L} \uplus \mathcal{R}$$
$$s \mapsto \begin{cases} \text{state in which } M \text{ exits } w_1w_2\cdots w_n \text{ when started} & \text{if } s \in \mathcal{L}; \\ \text{in state } s \text{ scanning } w_1, & \\ \text{state in which } M \text{ exits } w_1w_2\cdots w_n \text{ when started} & \text{if } s \in \mathcal{R}. \\ \text{in state } s \text{ scanning } w_n, & \end{cases}$$

The base case is given by $\delta_w(s)s$ if $|w|0$, and $\delta_w(s)\delta(s,w)$ if $|w|1$. The induction step divides into two similar cases according to whether $s \in \mathcal{L}$ or $s \in \mathcal{R}$. Let $u \in \Sigma^+$ and $a \in \Sigma$. We only illustrate the case $s \in \mathcal{L}$, namely M entering ua from the left. The case breaks off into two subcases according to whether M eventually falls off ua to the left ($\delta_u(s_k) \in \mathcal{R}$) or to the right ($\delta_a(l_k) \in \mathcal{L}$):

$$\delta_{ua}(s) \begin{cases} \delta_u(s_k) \text{ if } (\exists k \geq 0)(\exists s_0 s)(\exists l_1, l_2, \ldots, l_k \in \mathcal{L})(\exists s_1, s_2, \ldots, s_k, \delta_u(s_k) \in \\ \qquad \mathcal{R}) \text{ such that } l_i\delta_u(s_{i-1}) \text{ and } s_i\delta_a(l_i) \text{ for } 1 \leq i \leq k; \\ \delta_a(l_k) \text{ if } (\exists k \geq 0)(\exists l_0\delta_u(s), l_1, l_2, \ldots, l_k, \delta_a(l_k) \in \mathcal{L})(\exists r_1, r_2, \ldots, r_k \in \\ \qquad \mathcal{R}) \text{ such that } r_i\delta_a(l_{i-1}) \text{ and } l_i\delta_u(r_i) \text{ for } 1 \leq i \leq k. \end{cases}$$

Finally we define the *behavior monoid* $\mathcal{B}(M)$ of M to be the monoid $\{\delta_w \mid w \in \Sigma^*\}$ under composition. M is a $\mathbf{V}$*-machine* iff $\mathcal{B}(M) \in \mathbf{V}$.

The *output of* M on input $ww_1w_2\cdots w_n$ is $\lambda(s_1,w_1)\lambda(s_2,w_2)\cdots\lambda(s_n,w_n)$, where s_i is the state entered by M as it lands on w_i for the last time when started from its initial configuration on w (when $|w|0$ the output is the empty string).

3 Results

Here we present our different results which together prove Theorem 1. First we state that for all varieties closed under reversal, translations, bisequential functions, bimachines, and single-valued nondeterministic transducers yield the same class of functions. This proves the equivalence of the first three statements of Theorem 1, since group varieties and $\mathbf{A}$ are closed under reversal.

Theorem 4. *The following are equivalent when* $\mathbf{V}$ *is a monoid variety closed under reversal:*

(i) f *is a* $\mathbf{V}$*-translation.*
(ii) f *is a* $\mathbf{V}$*-bisequential function.*
(iii) There is a sequential $\mathbf{V}$*-bimachine that computes* f*.*
(iv) There is a single-valued nondeterministic $\mathbf{V}$*-transducer that computes* f*.*

The proof, which can be found in the full version of this paper [MSTV00], consists of a sequence of simulations among the different models that preserve monoid identities in the sense of Sect. 2. It even turns out that equivalence of (i) and (ii) holds for all varieties of monoids. Here, we only present the following implication:

Lemma 5. *Let $\varphi_1, \ldots, \varphi_t$ be* $\mathbf{V}$*-formulas. Then there is a* $\mathbf{V}$*-bisequential function f such that $[\varphi_1, \ldots, \varphi_t]f$.*

Proof. We first assume that $t1$, i.e. that the image of the translation is over a two-letter alphabet, which we take to be $\{Y, N\}$. Initially, let $\varphi\varphi_1$ be a formula over some alphabet Σ of the form $\varphi(x)Q_A y\psi(x,y)$, where A is a $\mathbf{V}$-language and ψ is quantifier-free. Let α be the congruence relation that defines $A \subseteq \Gamma^*$, i.e., $\Gamma^*/\alpha \in \mathbf{V}$. For simplicity, we assume that $\Gamma\{1,0\}$; the case of non-binary alphabets is treated similarly.

For every letter $a \in \Sigma$, we define from ψ three formulas $\psi_a^<$, ψ_a and $\psi_a^>$ as follows:

	replace $C_a(x)$ by	$C_b(x)$, $b \neq a$, by	$y < x$ by	yx by	$x < y$ by
To get $\psi_a^<$	true,	false,	true,	false,	false.
To get ψ_a	true,	false,	false,	true,	false.
To get $\psi_a^>$	true,	false,	false,	false,	true.

Observe that all these formulas have the only free variable y, since x no longer appears. Moreover, for each $a \in \Sigma$, we write ψ_a for the *constant* formula obtained by further evaluating ψ_a with the knowledge that $C_a(y)$. Now recall that the ith symbol in $[\varphi](w_1 \ldots w_n)$ is equal to Y iff

$$\psi(i,1)\ \psi(i,2)\ \cdots\ \psi(i,n) \in A, \tag{1}$$

where for convenience we wrote $\psi(i,j)$ for the zero-one truth value of $\psi(i,j)$ evaluated on $w_1 \ldots w_n$. But the $\{0,1\}$-word appearing in (1) is precisely the $\{0,1\}$-word

$$\psi_{w_i}^<(1)\ \ \psi_{w_i}^<(2)\ \ \cdots\ \ \psi_{w_i}^<(i-1)\ \ \psi_{w_i}\ \ \psi_{w_i}^>(i+1)\ \ \cdots\ \ \psi_{w_i}^>(n-1)\ \ \psi_{w_i}^>(n). \tag{2}$$

Fortunately, every ψ_a^R (for $a \in \Sigma$, $R \in \{<,>\}$) defines a length-preserving homomorphism, also denoted $\psi_a^R \colon \Sigma^* \to \{0,1\}^*$. So let a congruence $\sim$ on Σ^* be defined as follows: $u \sim v$ iff $\psi_a^R(u)\ \alpha\ \psi_a^R(v)$ holds for each $a \in \Sigma$ and for each $R \in \{<,>\}$. This is a $\mathbf{V}$-congruence, being the intersection of finitely many $\mathbf{V}$-congruences (indeed each separate homomorphism ψ_a^R defines a congruence whose induced monoid is the image of a submonoid of $\{0,1\}^*/\alpha$). The $\mathbf{V}$-bisequential function $f(\sim, m)$ therefore computes $[\varphi]$, where

$$m \colon (\Sigma^*/\!\sim) \times (\Sigma^*/\!\sim) \times \Sigma \to \{Y, N\}$$
$$([u]_\sim, [v]_\sim, a) \mapsto \begin{cases} Y & \text{if } [\psi_a^<(u)]_\alpha \circ [\psi_a]_\alpha \circ [\psi_a^>(v)]_\alpha \subseteq A, \\ N & \text{otherwise.} \end{cases}$$

Here $\circ$ computes in $\{0,1\}^*/\alpha$, and m is well defined because $w_1 \sim w_2$ implies that $\psi_a^R(w_1)\ \alpha\ \psi_a^R(w_2)$.

Now if φ is a Boolean combination of formulas, we consider the homomorphisms defined by all subformulas of φ of the form $Q_A y\psi(x,y)$ for quantifier-free

ψ. In the case of several formulas $[\varphi_1, \ldots, \varphi_t]$ instead of one formula $[\varphi]$, we again have to deal with more homomorphisms as above. In both cases, the definition of m becomes more complicated, but the congruence $\sim$ remains an intersection of constantly many $\mathbf{V}$-congruences. Hence we still obtain a $\mathbf{V}$-bisequential f. $\square$

When are 2-way machines equivalent to the models of Theorem 4? In the next lemma, we show that a 2-way $\mathbf{V}$-automaton can always be simulated by a $\mathbf{V}$-bimachine. The converse, however, requires additional assumptions about $\mathbf{V}$. Lemma 7 deals with the variety of all aperiodic monoids and the variety of all groups. The construction that is used in the proof of Lemma 7 results in particularly simple 2-way automata if $\mathbf{V}$ is an arbitrary group varity. This is stated as Corollary 8.

Lemma 6. *Let $M(\mathcal{L} \uplus \mathcal{R}, \Sigma, \Gamma, \delta, \lambda, l_0, F)$ be a two-way $\mathbf{V}$-automaton with output. Then there exists a $\mathbf{V}$-bimachine (M_1, M_2, g) that computes T_M.*

Proof. Given M, we first show how to construct automata $M_1(S_1, s_1, \Sigma, \delta_1)$, $M_2(S_2, s_2, \Sigma, \delta_2)$ and function g. Let $\circ$ denote multiplication in $\mathcal{B}(M)$ and let e be the corresponding identity.

We define M_1 as by $S_1\mathcal{B}(M)$, s_1e, and for each $f \in \mathcal{B}(M), a \in \Sigma$, we set $\delta_1(f, a)f \circ \delta_a$. Intuitively, after reading $w_1 \cdots w_{i-1}$ the automaton M_1 has computed the function $\delta_{w_1 \cdots w_{i-1}}$.

The definition of M_2 is analogous: $S_2\mathcal{B}(M)$, s_2e, and for each $f \in \mathcal{B}(M), a \in \Sigma$, we set $\delta_2(f, a)\delta_a \circ f$.

It is easy to show that, for each string $ww_1 \cdots w_n$ and each position $i \in \{1, \ldots, n\}$, we obtain $\hat{\delta}_1(s_1, w_1 \cdots w_{i-1})\delta_{w_1 \cdots w_{i-1}}$ and $\hat{\delta}_2(s_2, w_n \cdots w_{i+1})\delta_{w_{i+1} \cdots w_n}$. By combining this information with w_i and the fixed behavior of M on the endmarkers, the sequence of states that M takes at position i can be determined. From the very definition of the output of a two-way automaton it follows that a function g as claimed exists and can be obtained directly from the definition of M.

Next we show that all monoid identities that hold in M hold in M_1 and M_2 as well. Let $u, v \in \Sigma^*$ be such that $\delta_u\delta_v$. We even prove the stronger claims that $\hat{\delta}_1(\cdot, u)\hat{\delta}_1(\cdot, v)$ and $\hat{\delta}_2(\cdot, u^R)\hat{\delta}_2(\cdot, v^R)$.

From the definition of M_1 we obtain for all $f \in \mathcal{B}(M), w \in \Sigma$, that $\hat{\delta}_1(f, w)f \circ \delta_w$, which immediately yields the first claim. From the definition of M_2 we conclude that for each $f \in \mathcal{B}(M)$ and $w \in \Sigma^*$, we obtain $\hat{\delta}_2(f, w)\delta_{w^R} \circ f$, where w^R denotes the reversal of the word w. This immediately implies the second claim. Hence we showed that M_1 and M_2 are $\mathbf{V}$-DFAs. $\square$

Next we turn a converse of this lemma. An important tool here is a result of Hopcroft and Ullman [HU67], showing how a two-way automaton can, for each symbol in its input, determine the states of one left-to-right and one right-to-left automaton, when they reach the corresponding input symbol. The next lemma shows that this result carries over to the case of restricted types of automata.

Lemma 7. *Let (M_1, M_2, g) be a sequential bimachine. Then there exists a 2-way automaton $M(\mathcal{L} \uplus \mathcal{R}, \Sigma, \Gamma, \delta, \lambda, l_0, \emptyset)$ which computes the same function as (M_1, M_2, g) and has the following property: If $\hat{\delta}_1(\cdot, u)\hat{\delta}_1(\cdot, v)$ and $\hat{\delta}_2(\cdot, u^R)\hat{\delta}_2(\cdot, v^R)$ and $uvvu$ then*

$$\delta_{uu}\delta_{uv}\delta_{vu}\delta_{vv}.$$

Proof. We sketch the idea of the proof, and we refer the reader to [MSTV00] for the full details. Let $M'(M_1, M_2, g)$, $M_1(Q_1, s_1, \Sigma, \delta_1)$, and $M_2(Q_2, s_2, \Sigma, \delta_2)$.

Informally, the behavior of the automaton M on input $ww_1 \cdots w_n$ is split into two phases. Roughly, the first phase consists of a complete left-to-right scan in which M simulates M_1 and the second phase consists of n subcomputations each of which is a right-to-left movement followed by a (possibly empty) left-to-right movement. In the second phase M simulates the behavior of M_2 (which is easy) and keeps track of the states of M_1 (which is more complicated).

In more detail, each of the n subcomputations starts from a situation in which the automaton is at a position i of its input and knows $p_i\hat{\delta}_1(s_1, w_1 \cdots w_i)$ and $q_i\hat{\delta}_2(s_2, w_n \cdots w_{i+1})$ and ends in the corresponding situation at position $i-1$.

The obvious problem is how to compute $p_{i-1}\hat{\delta}_1(s_1, w_1 \cdots w_{i-1})$. If there is only one state p of Q_1 with $\delta_1(p, w_i)p_i$ then, of course $p_{i-1}p$. Otherwise M proceeds as follows. It moves to the left and maintains, for each state p with $\delta_1(p, w_i)p_i$, a set P_p of states. At a position $j < i$, P_p contains all states from which M_1 can reach p by reading $w_{j+1} \cdots w_{i-1}$. It is easy to see that, for each j, these sets are pairwise disjoint. This process ends in one of the following two situations. Either all but one of the sets P_p become empty or M reaches the left delimiter. In either case M can easily conclude p_{i-1} (in the latter case it is the state p for which P_p contains s_1). Now it only needs to find its way back to position i. In order to do so, it goes one step to the right and chooses one state from P_p and one state from a nonempty $P_{p'}$ at that position (these two sets were remembered by M). Then it simulates the behavior of M_1 starting from these two states until the two computations flow together into a single state. At this time M has reached position i again and now it also knows p_{i-1}, as this is given by the outcome of the simulation of M_1 which started from the correct state. It has remembered q_i during the whole subcomputation, hence it can retrieve the output symbol of M at position i. Then it starts the next subcomputation from position $i-1$. It is this last step of a subcomputation where M actually simulates the behavior of M_2.

The formal definition of M and the (technically involved) proof that it has the properties claimed in the lemma can be found in [MSTV00]. □

If $\mathbf{V}$ is a variety of groups then the construction in the proof of Lemma 7 leads to a very simple 2-way automaton. The simple reason is that, for such $\mathbf{V}$, $\mathbf{V}$-automata have injective transition functions. Hence, for each state p of the automaton M_1 and each symbol σ there is exactly one state p' such that $\delta_1(p', \sigma)p$. Therefore, M can always deduce p_{i-1} from p_i by one step to the left. It follows directly that all word equations that hold in M_1 and (reversed) in M_2

also hold in M and we can conclude the following corollary which considerably strengthens Lemma 7 for group varieties.

Corollary 8. *Let (M_1, M_2, g) be a sequential $\mathbf{V}$-bimachine computing a function f, where $\mathbf{V}$ an arbitrary group variety. Then f is computable by a 2-way $\mathbf{V}$-automaton.*

Theorem 9. *Let $\mathbf{V}$ be the variety $\mathbf{A}$ or an arbitrary variety of groups. A function is computed by a sequential $\mathbf{V}$-bimachine iff it is computed by a 2-way $\mathbf{V}$-machine.*

Proof. Lemma 6 shows that a sequential $\mathbf{V}$-bimachine can simulate a 2-way $\mathbf{V}$-machine. For the converse, there are two cases. The first case is when $\mathbf{V}$ is the variety $\mathbf{A}$ of all aperiodic monoids. Let a $\mathbf{V}$-bimachine (M_1, M_2, g) be simulated by the 2-way automaton M promised by Lemma 7. Since M_1 and M_2 are aperiodic, there exists $n \geq 0$ such that for all words w, $\hat{\delta}_i(\cdot, w^n)\hat{\delta}_i(\cdot, w^{n+1})$, $i1, 2$. But then $\mathcal{B}(M) \in \mathbf{A}$ because Lemma 7 applies with uw^n and vw^{n+1}:

$$\forall w, (\delta_w)^{2n}\delta_{w^{2n}}\delta_{uu}\delta_{uv}\delta_{w^{2n+1}}(\delta_w)^{2n+1},$$

where $(\delta_x)^i$ is the i-fold composition of δ_x with itself.

The second case is when $\mathbf{V}$ is an arbitrary group variety. In this case the result follows from Corollary 8. □

4 Conclusion

We have characterized FO-translations in several natural ways, and we have extended the characterization to handle group varieties. The strong equivalences obtained, with the exception of the 2-way automaton characterization, extend to any monoid variety $\mathbf{V}$ closed under reversal.

We believe that Theorem 1 generalizes to many more varieties. The hurdle lies in the fine details required to ensure that a 2-way automaton simulating any one of the other models preserves enough of the algebraic structure. We feel here that we have not exhausted all the tricks. For example, crossing sequence arguments may be applicable to improve the construction and its analysis.

Our careful definition of what constitutes a 2-way $\mathbf{V}$-automaton opens the way for an algebraic treatment of 2-way automata. The translations studied here are akin to the maps underlying the wreath product and the block product constructions. This suggests approaches to handle the nesting of monoidal quantifiers in the logical framework, or equivalently to handle the series connections of 2-way automata. This emerges as a promising avenue for future research.

Acknowledgment

For helpful comments, we are grateful to Heinz Schmitz.

References

BCST92. D. A. Mix Barrington, K. Compton, H. Straubing, and D. Thérien. Regular languages in NC^1. *Journal of Computer and System Sciences*, 44:478–499, 1992.

BIS90. D. A. Mix Barrington, N. Immerman, and H. Straubing. On uniformity within NC^1. *Journal of Computer and System Sciences*, 41:274–306, 1990.

Büc60. J. R. Büchi. Weak second-order arithmetic and finite automata. *Math. Logik. Grund. Math.*, 6:66–92, 1960.

Eil76a. S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, New York, 1976.

Eil76b. S. Eilenberg. *Automata, Languages, and Machines*, volume B. Academic Press, New York, 1976.

HU67. J. E. Hopcroft and J. E. Ullman. An approach to a unified theory of automata. In *Proceedings 8th Symposium on Switching and Automata Theory*, pages 140–147. IEEE Computer Society Press, 1967.

HU79. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, MA, 1979.

Imm99. N. Immerman. *Descriptive Complexity.* Graduate Texts in Computer Science. Springer Verlag, New York, 1999.

LMSV99. C. Lautemann, P. McKenzie, T. Schwentick, and H. Vollmer. The descriptive complexity approach to LOGCFL. In *Proceedings 16th Symposium on Theoretical Aspects of Computer Science*, volume 1563 of *Lecture Notes in Computer Science*, pages 444–454. Springer Verlag, 1999.

MP71. R. McNaughton and S. Papert. *Counter-Free Automata.* MIT Press, 1971.

MSTV00. P. McKenzie, T. Schwentick, D. Thérien, and H. Vollmer. The many faces of a translation. Technical Report 248, Institut für Informatik, Universität Würzburg, 2000.
URL: http://www.informatik.uni-wuerzburg.de/reports/tr.html.

RS59. M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res.*, 3(2):115–125, 1959.

Sch65. M. P. Schützenberger. On finite monoids having only trivial subgroups. *Information & Control*, 8:190–194, 1965.

She59. J. C. Sheperdson. The reduction of two-way automata to one-way automata. *IBM J. Res.*, 3(2):198–200, 1959.

Str94. H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, Boston, 1994.

STT88. H. Straubing, D. Thérien, and W. Thomas. Regular languages defined with generalized quantifiers. In *Proceedings 15th International Colloqium on Automata, Languages and Programming*, volume 317 of *Lecture Notes in Computer Science*, pages 561–575, Berlin Heidelberg, 1988. Springer Verlag.

Tra61. B. A. Trakhtenbrot. Finite automata and logic of monadic predicates. *Doklady Akademii Nauk SSSR*, 140:326–329, 1961. In Russian.

Gales and the Constructive Dimension of Individual Sequences

Jack H. Lutz *

Department of Computer Science
Iowa State University
Ames, IA 50011, USA
lutz@cs.iastate.edu

Abstract. A constructive version of Hausdorff dimension is developed and used to assign to every individual infinite binary sequence A a *constructive dimension*, which is a real number $cdim(A)$ in the interval $[0, 1]$. Sequences that are random (in the sense of Martin-Löf) have constructive dimension 1, while sequences that are decidable, r.e., or co-r.e. have constructive dimension 0. It is shown that for every Δ_2^0-computable real number α in $[0, 1]$ there is a Δ_2^0 sequence A such that $cdim(A) = \alpha$. Every sequence's constructive dimension is shown to be bounded above and below by the limit supremum and limit infimum, respectively, of the average Kolmogorov complexity of the sequence's first n bits. Every sequence that is random relative to a computable sequence of rational biases that converge to a real number β in $(0, 1)$ is shown to have constructive dimension $\mathcal{H}(\beta)$, the binary entropy of β.
Constructive dimension is based on *constructive gales*, which are a natural generalization of the constructive martingales used in the theory of random sequences.

Keywords: algorithmic information, computability, constructive dimension, gales, Hausdorff dimension, Kolmogorov complexity, martingales, randomness.

1 Introduction

One of the most dramatic achievements of the theory of computing was Martin-Löf's 1966 use of constructive measure theory to give the first satisfactory definition of the randomness of individual infinite binary sequences [16]. The search for such a definition had been a major object of early twentieth-century research on the foundations of probability, but a rigorous mathematical formulation had proven so elusive that the search had been all but abandoned more than two decades earlier. Martin-Löf's definition says precisely which infinite binary sequences are random and which are not. The definition is probabilistically convincing in that it requires each random sequence to pass every algorithmically implementable statistical test of randomness. The definition is also robust in that

* This work was supported in part by National Science Foundation Grant 9610461.

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 902–913, 2000.
© Springer-Verlag Berlin Heidelberg 2000

subsequent definitions by Schnorr [22, 23, 24], Levin [11], Chaitin [4], Solovay [28], and Shen' [25, 26], using a variety of different approaches, all define exactly the same sequences to be random. It is noteworthy that all these approaches, like Martin-Löf's, make essential use of the theory of computing.

One useful characterization of random sequences is that they are those sequences that have maximal algorithmic information content. Specifically, if $K(A[0..n-1])$ denotes the Kolomogorov complexity (algorithmic information content) of the first n bits of an infinite binary sequence A, then Levin [11] and Chaitin [4] have shown that A is random if and only if there is a constant c such that for all n, $K(A[0..n-1]) \geq n - c$. Indeed Kolmogorov [8] developed what is now called $C(x)$, the "plain Kolmogorov complexity," in order to formulate such a definition of randomness, and Martin-Löf, who was then visiting Kolmogorov, was motivated by this idea when he defined randomness. (The quantity $C(x)$ was also developed independently by Solomonoff [27] and Chaitin [2, 3].) Martin-Löf [17] subsequently proved that $C(x)$ cannot be used to characterize randomness, and Levin [11] and Chaitin [4] introduced a technical modification of $C(x)$, now called $K(x)$, the "Kolmogorov complexity," in order to prove the above characterization of random sequences. Schnorr [24] proved a similar characterization in terms of another variant, called the "monotone Kolmogorov complexity."

One conclusion to be drawn from these characterizations is that the definition of random sequences distinguishes those sequences that have maximal algorithmic information content from those that do not. It offers no quantitative classification of the sequences that have less than maximal algorithmic information content. From a technical point of view, this aspect of the definition arises from its use of constructive measure, which is an algorithmic effectivization of classical Lebesgue measure. Specifically, an infinite binary sequence A is random if the singleton set $\{A\}$ does not have constructive measure 0, and is nonrandom if $\{A\}$ does have constructive measure 0. Neither Lebesgue measure nor constructive measure offers quantitative distinctions among measure 0 sets.

In 1919, Hausdorff [6] augmented classical Lebesgue measure theory with a theory of dimension. This theory assigns to every subset X of a given metric space a real number $dim_H(X)$, which is now called the *Hausdorff dimension* of X. In this paper we are interested in the case where the metric space is the Cantor space $\mathbf{C}$, consisting of all infinite binary sequences. In this case, the Hausdorff dimension of a set $X \subseteq \mathbf{C}$ (which is defined precisely in section 3 below) is a real number $dim_H(X) \in [0, 1]$. The Hausdorff dimension is monotone, with $dim_H(\emptyset) = 0$ and $dim_H(\mathbf{C}) = 1$. Moreover, if $dim_H(X) < dim_H(\mathbf{C})$, then X is a measure 0 subset of $\mathbf{C}$. Hausdorff dimension thus offers a quantitative classification of measure 0 sets. Moreover, Ryabko [19, 20, 21] Staiger [29, 30], and Cai and Hartmanis [1] have all proven results establishing quantitative relationships between Hausdorff dimension and Kolmogorov complexity.

Just as Hausdorff [6] augmented Lebesgue measure with a theory of dimension, this paper augments the theory of individual random sequences with a theory of the constructive dimension of individual sequences. Specifically, we develop a constructive version of Hausdorff dimension and use this to assign ev-

ery sequence $A \in \mathbf{C}$ a *constructive dimension* $cdim(A) \in [0,1]$. Sequences that are random have constructive dimension 1, while sequences that are decidable, r.e., or co-r.e. have constructive dimension 0. For every real number $\alpha \in [0,1]$ there is a sequence A such that $cdim(A) = \alpha$. Moreover, if α is Δ_2^0-computable, then there is a Δ_2^0 sequence A such that $cdim(A) = \alpha$. (This generalizes the well-known existence of Δ_2^0 sequences that are random.)

Regarding algorithmic information content, we prove that for every sequence $A \in \mathbf{C}$,

$$\liminf_{n\to\infty} \frac{K(A[0..n-1])}{n} \leq dim(A) \leq \limsup_{n\to\infty} \frac{K(A[0..n-1])}{n}.$$

This justifies the intuition that the constructive dimension of a sequence is a measure of its *algorithmic information density.*

We also relate constructive dimension to Shannon entropy. If $\overrightarrow{\beta} = (\beta_0, \beta_1, \dots)$ is any computable sequence of rational numbers $\beta_i \in [0,1]$ that converge to a real number $\beta \in (0,1)$ (which must therefore be Δ_2^0-computable), we show that every sequence that is random relative to $\overrightarrow{\beta}$ has constructive dimension $\mathcal{H}(\beta)$, the binary entropy of β.

Our development of constructive dimension is based on *gales*, which are natural generalizations of the constructive martingales used by Schnorr [22, 23, 24] to characterize randomness. In a recent paper [15] we have shown that gales can be used to characterize the classical Hausdorff dimension, and that resource-bounded gales can be used to define dimension in complexity classes. In the present paper we use constructive (lower semicomputable) gales to develop constructive dimension. Constructive dimension differs markedly from both classical Hausdorff dimension and the resource-bounded dimension developed in [15], primarily due to the existence of gales that are optimal. These optimal gales, defined in section 4, are analogous to universal tests of randomness in the theory of random sequences.

2 Preliminaries

We work in the Cantor space $\mathbf{C}$ consisting of all infinite binary sequences. The n-bit prefix of a sequence $A \in \mathbf{C}$ is the string $A[0..n-1] \in \{0,1\}^*$ consisting of the first n bits of A. We say that a string $w \in \{0,1\}^*$ is a *prefix* of a sequence $A \in \mathbf{C}$, and we write $w \sqsubseteq A$ if $A[0..|w|-1] = w$. The *cylinder generated by* a string $w \in \{0,1\}^*$ is $\mathbf{C}_w = \{A \in \mathbf{C} \mid w \sqsubseteq A\}$. Note that $\mathbf{C}_\lambda = \mathbf{C}$, where λ is the empty string.

Definition. A *probability measure* on $\mathbf{C}$ is a function $\nu : \{0,1\}^* \to [0,1]$ with the following two properties.

(i) $\nu(\lambda) = 1$.
(ii) For all $w \in \{0,1\}^*$, $\nu(w) = \nu(w0) + \nu(w1)$.

Intuitively, $\nu(w)$ is the probability that $A \in \mathbf{C}_w$ when $A \in \mathbf{C}$ is "chosen according to the probability measure ν."

Definition. A *bias sequence* is a sequence $\overrightarrow{\beta} = (\beta_0, \beta_1, \beta_2, \dots)$, where each $\beta_i \in [0,1]$. A bias sequence $\overrightarrow{\beta} = (\beta_0, \beta_1, \dots)$ is *strongly positive* if there exists $\delta > 0$ such that for all $i \in \mathbb{N}$, $\delta \leq \beta_i \leq 1 - \delta$.

Definition. If $\overrightarrow{\beta}$ is a bias sequence, then the $\overrightarrow{\beta}$*-coin-toss probability measure* on $\mathbf{C}$ is the probability measure

$$\mu_{\overrightarrow{\beta}} : \{0,1\}^* \to [0,1]$$

$$\mu_{\overrightarrow{\beta}}(w) = \prod_{i=0}^{|w|-1} \beta_i(w),$$

where $\beta_i(w) = (2\beta_i - 1)w[i] + (1 - \beta_i)$, i.e., $\beta_i(w) =$ if $w[i] = 1$ then β_i else $1 - \beta_i$.

Note that $\mu_{\overrightarrow{\beta}}(w)$ is the probability that $A \in \mathbf{C}_w$ when $A \in \mathbf{C}$ is chosen according to a random experiment in which for each i, independently of all other j, the ith bit of A is decided by tossing a 0/1-valued coin whose probability of 1 is β_i.

Definition. If $\beta \in [0,1]$, then the β*-coin-toss probability measure* on $\mathbf{C}$ is the probability measure $\mu_\beta = \mu_{\overrightarrow{\beta}}$, where $\overrightarrow{\beta} = (\beta, \beta, \beta, \dots)$.

Definition. The *uniform probability measure* on $\mathbf{C}$ is the probability measure μ defined by $\mu(w) = 2^{-|w|}$ for all $w \in \{0,1\}^*$. (Note that $\mu = \mu_{\frac{1}{2}}$.)

We use several conditions involving the computability of real numbers and real-valued functions in this paper.

Definition. Let $\alpha \in \mathbb{R}$,

1. α is *computable* if there is a computable function $f : \mathbb{N} \to \mathbb{Q}$ such that for all $r \in \mathbb{N}$, $|f(r) - \alpha| \leq 2^{-r}$.
2. α is *lower semicomputable* if its Dedekind left cut $left(\alpha) = \{s \in \mathbb{Q} \mid s < \alpha\}$ is recursively enumerable.
3. α is Δ^0_2*-computable* if α is computable relative to the halting oracle.

It is well known and easy to verify that every computable real is lower semicomputable, that every lower semicomputable real is Δ_2^0-computable, and that the converses of these statements do not hold.

Definition. Let $f : D \to \mathbb{R}$, where D is some discrete domain such as $\{0,1\}^*$ or $\mathbb{N}$. Then f is *lower semicomputable* if its lower graph

$$Graph^-(f) = \{(x,s) \in D \times \mathbb{Q} \mid s < f(x)\}$$

is recursively enumerable.

A *prefix set* is a set $B \subseteq \{0,1\}^*$ such that no element of B is a prefix of any other element of B.

The reader is referred to the text by Li and Vitányi [12] for the definition and basic properties of the Kolmogorov complexity $K(x)$, defined for strings $x \in \{0,1\}^*$. Falconer [5] provides a good overview of Hausdorff dimension.

3 Gales and Hausdorff Dimension

In this section we define gales and supergales and use these to define classical Hausdorff dimension in the Cantor space $\mathbf{C}$. Our definitions are slightly more general than those in [15] because here we need to define gales and supergales relative to an arbitrary (not necessarily uniform) probability measure on $\mathbf{C}$.

Definition. Let ν be a probability measure on $\mathbf{C}$, and let $s \in [0,\infty)$.

1. A *ν-s-supergale* is a function $d : \{0,1\}^* \to [0,\infty)$ that satisfies the condition

$$d(w)\nu(w)^s \geq d(w0)\nu(w0)^s + d(w1)\nu(w1)^s \qquad (*)$$

   for all $w \in \{0,1\}^*$.
2. A *ν-s-gale* is a ν-s-supergale that satisfies $(*)$ with equality for all $w \in \{0,1\}^*$.
3. A *ν-supermartingale* is a ν-1-supergale.
4. A *ν-martingale* is a ν-1-gale.
5. An *s-supergale* is a μ-s-supergale.
6. An *s-gale* is a μ-s-gale.
7. A *supermartingale* is a 1-supergale.
8. A *martingale* is a 1-gale.

The following obvious but useful observation shows how gales and supergales are affected by variation of the parameter s.

Observation 3.1 *Let ν be a probability measure on $\mathbf{C}$, let s, $s' \in [0,\infty)$, and let $d, d' : \{0,1\}^* \to [0,\infty)$. Assume that*

$$d(w)\nu(w)^s = d'(w)\nu(w)^{s'}$$

for all $w \in \{0,1\}^*$.

1. *d is a ν-s-supergale if and only if d' is a ν-s'-supergale.*
2. *d is a ν-s-gale if and only if d' is a ν-s'-gale.*

For example, Observation 3.1 implies that a function $d : \{0,1\}^* \to [0,\infty)$ is an s-gale if and only if the function $d' : \{0,1\}^* \to [0,\infty)$ defined by $d'(w) = 2^{(1-s)|w|}d(w)$ is a martingale.

The following useful lemma is a generalization of Kraft's inequality.

Lemma 3.2. *Let d be a ν-s-supergale, where ν is a probability measure on $\mathbf{C}$ and $s \in [0,\infty)$. Then for all $w \in \{0,1\}^*$ and all prefix sets $B \subseteq \{0,1\}^*$,*

$$\sum_{u \in B} d(wu)\nu(wu)^s \leq d(w)\nu(w)^s.$$

Definition. Let d be a ν-s-supergale, where ν is a probability measure on $\mathbf{C}$ and $s \in [0,\infty)$.

1. We say that d *succeeds* on a sequence $A \in \mathbf{C}$ if $\limsup_{n\to\infty} d(A[0..n-1]) = \infty$.
2. The *success set* of d is $S^\infty[d] = \{A \in \mathbf{C} \mid d \text{ succeeds on } A\}$.

We now show how to use the success sets of gales and supergales to define Hausdorff dimension.

Notation. Let $X \subseteq \mathbf{C}$.

1. $\mathcal{G}(X)$ is the set of all $s \in [0,\infty)$ such that there is an s-gale d for which $X \subseteq S^\infty[d]$.
2. $\widehat{\mathcal{G}}(X)$ is the set of all $s \in [0,\infty)$ such that there is an s-supergale d for which $X \subseteq S^\infty[d]$.

Lemma 3.3. *For all $X \subseteq \mathbf{C}$, $\mathcal{G}(X) = \widehat{\mathcal{G}}(X)$.*

It was shown in [15] that the following definition is equivalent to the classical definition of Hausdorff dimension in $\mathbf{C}$.

Definition. The *Hausdorff dimension* of a set $X \subseteq \mathbf{C}$ is $dim_H(X) = \inf \mathcal{G}(X)$.

Note that by Lemma 3.3 we could equivalently use $\widehat{\mathcal{G}}(X)$ in place of $\mathcal{G}(X)$ in the above definition.

4 Constructive Dimension of Individual Sequences

In this section we constructivize the above definition of Hausdorff dimension and use this to define the constructive dimensions of individual sequences. We then develop some fundamental properties of constructive dimension.

Terminology. A ν-s-supergale is *constructive* if it is lower semicomputable.

Notation. Let $X \subseteq \mathbf{C}$.

1. $\mathcal{G}_{\text{constr}}(X)$ is the set of all $s \in [0, \infty)$ such that there is a constructive s-gale d for which $X \subseteq S^\infty[d]$.
2. $\widehat{\mathcal{G}}_{\text{constr}}(X)$ is the set of all $s \in [0, \infty)$ such that there is constructive s-supergale d for which $X \subseteq S^\infty[d]$.

The following constructive analog of Lemma 3.3 is proven using Observation 3.1 and known properties of constructive martingales.

Lemma 4.1. *For all $X \in \mathbf{C}$, $\mathcal{G}_{constr}(X)$ is a dense subset of $\widehat{\mathcal{G}}_{constr}(X)$.*

In light of the foregoing, the following definition is quite natural.

Definition. The *constructive dimension* of a set $X \subseteq \mathbf{C}$ is $cdim(X) = \inf \mathcal{G}_{\text{constr}}(X)$.

By Lemma 4.1, we also have $cdim(X) = \inf \widehat{\mathcal{G}}_{\text{constr}}(X)$.

Recall that a set $X \subseteq \mathbf{C}$ has *constructive measure* 0 if there is a constructive martingale d such that $X \subseteq S^\infty[d]$. The following useful observations are clear.

Observations 4.2 *1. For all $X \subseteq Y \subseteq \mathbf{C}$, $cdim(X) \leq cdim(Y)$.*
2. For all $X \subseteq \mathbf{C}$, $cdim(X) \geq dim_H(X)$.
3. $cdim(\mathbf{C}) = 1$.
4. For all $X \subseteq \mathbf{C}$, if $cdim(X) < 1$, then X has constructive measure 0.

We now introduce the crucial concept of optimal constructive gales.

Definition. Let $s \in [0, \infty)$. A constructive s-gale d is *optimal* if for every constructive s-gale d' there is a constant $\epsilon > 0$ such that for all $w \in \{0,1\}^*$, $d(w) \geq \epsilon d'(w)$.

Schnorr [22, 23] has established the existence of an optimal constructive martingale, which we call $d^{(1)}$. For each $s \in [0, \infty)$, we define the function

$$d^{(s)} : \{0,1\}^* \to [0, \infty)$$

$$d^{(s)}(w) = 2^{(s-1)|w|} d^{(1)}(w).$$

The following important result follows immediately from Observation 3.1 and the optimality of $d^{(1)}$.

Theorem 4.3. *For every computable real number $s \in [0, \infty)$, the function $d^{(s)}$ is an optimal constructive s-gale.*

We now define the constructive dimensions of individual sequences.

Definition. The *constructive dimension* of a sequence $A \in \mathbf{C}$ is $cdim(A) = cdim(\{A\})$. For each $\alpha \in [0,1]$, we write $\mathrm{DIM}^{\alpha} = \{A \in \mathbf{C} \mid cdim(A) = \alpha\}$ and $\mathrm{DIM}^{\leq\alpha} = \{A \in \mathbf{C} \mid cdim(A) \leq \alpha\}$.

We first give a simple diagonalization proof that for every real number $\alpha \in [0,1]$, there is a sequence whose constructive dimension is α.

Lemma 4.4. *For all $\alpha \in [0,1]$, $\mathrm{DIM}^{\alpha} \neq \emptyset$.*

The constructive dimension of a set can be completely characterized in terms of the constructive dimensions of its elements in the following way.

Lemma 4.5. *For all $X \subseteq \mathbf{C}$, $cdim(X) = \sup_{A \in X} cdim(A)$.*

Lemma 4.5, which has no analog either in classical Hausdorff dimension or in the resource-bounded dimension developed in [15], depends crucially on Theorem 4.3. Lemma 4.5 yields an easy proof that constructive dimension has the following property (which is also a property of classical Hausdorff dimension).

Corollary 4.6. *For all $X_0, X_1, X_2, \ldots \subseteq \mathbf{C}$, $cdim(\cup_{k=0}^{\infty} X_k) = \sup_{k \in \mathbb{N}} cdim(X_k)$.*

Lemmas 4.4 and 4.5 also have the following consequence, which states that for each $\alpha \in [0,1]$, $\mathrm{DIM}^{\leq\alpha}$ is the largest set of dimension α.

Corollary 4.7. *For every real number $\alpha \in [0,1]$, the set $\mathrm{DIM}^{\leq\alpha}$ has the following two properties.*

1. *$cdim(\mathrm{DIM}^{\leq\alpha}) = \alpha$.*
2. *For all $X \subseteq \mathbf{C}$, if $cdim(X) \leq \alpha$, then $X \subseteq \mathrm{DIM}^{\leq\alpha}$.*

We also have the following.

Corollary 4.8. *For every real number $\alpha \in [0,1]$, $cdim(\mathrm{DIM}^{\alpha}) = \alpha$.*

Recall that a sequence $A \in \mathbf{C}$ is *random* if the singleton set $\{A\}$ does not have constructive measure 0. We write RAND for the set of all random sequences. The following is obvious.

Observation 4.9 $\mathrm{RAND} \subseteq \mathrm{DIM}^{1}$.

It is well known that there are no random sequences in the set $\Sigma_1^0 \cup \Pi_1^0$, consisting of all characteristic sets of r.e or co-r.e. sets. We now show that much more is true.

Lemma 4.10. $\Sigma_1^0 \cup \Pi_1^0 \subseteq \mathrm{DIM}^{0}$.

An important result in the theory of random sequences is the existence of random sequences in Δ_2^0. By Observation 4.9, this immediately implies the existence of Δ_2^0 sequences of constructive dimension 1. We would like to extend this result to other positive dimensions. We first note that, since Δ_2^0 is countable, there can only be Δ_2^0 sequences of countably many different constructive dimensions. We also note that the proof that $\mathrm{RAND} \cap \Delta_2^0 \neq \emptyset$ using Kreisel's Basis Lemma [10, 32, 18] and the fact that RAND is a Σ_2^0 class does not directly carry over to the present question because the class DIM^α does not appear to be Σ_2^0. (Note: Indeed, Terwijn [31] has very recently proven that it is not.) Instead, we first prove a dimension reduction theorem, which has independent interest, and then use this to derive the existence of Δ_2^0 sequences of constructive dimension α (for suitable α) from the existence of Δ_2^0 sequences that are random.

Define an *approximator* of a real number $\alpha \in [0,1]$ to be an ordered pair (a,b) of computable functions $a, b : \mathbb{N} \to \mathbb{Z}^+$ with the following properties.

(i) For all $n \in \mathbb{N}$, $a(n) \leq b(n)$.
(ii) $\lim_{n\to\infty} \frac{a(n)}{b(n)} = \alpha$.

It is well known and easy to see that a real number $\alpha \in [0,1]$ has an approximator if and only if it is Δ_2^0-computable. Moreover, every Δ_2^0-computable real number has an approximator (a,b) that is *nice* in the sense that if we let $\tilde{b}(k) = \sum_{n=0}^{k-1} b(n)$, then $b(k) = o(\tilde{b}(k))$ as $k \to \infty$.

Given an approximator (a,b) of a Δ_2^0-computable real number $\alpha \in [0,1]$, we define the (a,b)-*dilution function*

$$g_{(a,b)} : \mathbf{C} \to \mathbf{C}$$

as follows. Given $A \in \mathbf{C}$, if we write

$$A = w_0 w_1 w_2 \ldots ,$$

where $|w_n| = a(n)$ for each $n \in \mathbb{N}$, then

$$g_{(a,b)}(A) = w_0 0^{b(0)-a(0)} w_1 0^{b(1)-a(1)} \ldots .$$

Note that $g_{(a,b)}(A) \equiv_1 A$ for all $A \in \mathbf{C}$.

Theorem 4.11. *Let $\alpha \in [0,1]$ be Δ_2^0-computable, and let (a,b) be a nice approximator of α. Then for all $A \in \mathbf{C}$,*

$$cdim(g_{(a,b)}(A)) = \alpha \cdot cdim(A).$$

By Theorem 4.11, Observation 4.9, and the known existence of Δ_2^0 sequences that are random, we now have the following.

Theorem 4.12. *For every Δ_2^0-computable real number $\alpha \in [0,1]$, $\mathrm{DIM}^\alpha \cap \Delta_2^0 \neq \emptyset$, i.e., there is a Δ_2^0 sequence A such that $cdim(A) = \alpha$.*

Note that the proof of Theorem 4.12 via Theorem 4.11 yields even more, namely, that if $\alpha, \beta \in [0,1]$ are Δ^0_2-computable with $\alpha \geq \beta$, then every sequence in DIM^α is 1-equivalent to some sequence in DIM^β.

We now relate constructive dimension to Kolmogorov complexity.

Theorem 4.13. *For all $A \in \mathbf{C}$,*

$$\liminf_{n\to\infty} \frac{K(A[0..n-1])}{n} \leq cdim(A) \leq \limsup_{n\to\infty} \frac{K(A[0..n-1])}{n}.$$

Much of the technical content of the proof of Theorem 4.13 can be found in the investigations by Ryabko [19, 20, 21] and Staiger [29, 30] of the relationships between Kolmogorov complexity and classical Hausdorff dimension, and also in the calculation by Cai and Hartmanis [1] of the Hausdorff dimension of the graph of the average Kolmogorov complexities of real numbers.

Theorem 4.13 justifies the intuition that the constructive dimension of a sequence is its *algorithmic information density.*

Our last result on constructive dimension relates randomness over non-uniform distributions to constructive dimension. Recall that a sequence $A \in \mathbf{C}$ is *random relative to* a probability measure ν on $\mathbf{C}$ if there is no constructive ν-martingale d such that $A \in S^\infty[d]$. Given a bias sequence $\overrightarrow{\beta}$, we write $\mathrm{RAND}_{\overrightarrow{\beta}}$ for the set of all sequences that are random relative to the $\overrightarrow{\beta}$-coin-toss probability measure $\mu_{\overrightarrow{\beta}}$ defined in section 2. Recall also the binary entropy function

$$\mathcal{H}(\beta) = \beta \log \frac{1}{\beta} + (1-\beta) \log \frac{1}{1-\beta}$$

of Shannon information theory.

Theorem 4.14. *If $\overrightarrow{\beta}$ is a computable sequence of rational biases that converges to a real number $\beta \in (0,1)$, then*

$$\mathrm{RAND}_{\overrightarrow{\beta}} \subseteq \mathrm{DIM}^{\mathcal{H}(\beta)}.$$

Note that Observation 4.9 is exactly the case $\overrightarrow{\beta} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \dots)$ of Theorem 4.14. Note also that Theorem 4.14 can be used to give a second (albeit less informative) proof of Theorem 4.12.

Computable bias sequences that converge slowly to $\frac{1}{2}$ have played an important role in the investigation of stochasticity versus randomness. First, van Lambalgen [32] and, independently, Vovk [33] proved that if $\overrightarrow{\beta}$ is a bias sequence such that $\sum_{i=0}^\infty (\beta_i - \frac{1}{2})^2 = \infty$, then $\mathrm{RAND}_{\overrightarrow{\beta}} \cap \mathrm{RAND} = \emptyset$. Also, van Lambalgen [32] proved that if $\overrightarrow{\beta}$ is any computable bias sequence that converges to $\frac{1}{2}$, then every element of $\mathrm{RAND}_{\overrightarrow{\beta}}$ is Church-stochastic. Taking $\overrightarrow{\beta}$ to converge to $\frac{1}{2}$, but to do so slowly enough that $\sum_{i=0}^\infty (\beta_i - \frac{1}{2})^2 = \infty$ (e.g., $\beta_i = \frac{1}{2} + \frac{1}{\sqrt{i+2}}$), this gave a new proof that not every Church-stochastic sequence is random. More significantly, Shen′ [26] strengthened van Lambalgen's latter result by showing that if $\overrightarrow{\beta}$ is any computable bias sequence that converges to $\frac{1}{2}$, then every element

of $\mathrm{RAND}_{\overrightarrow{\beta}}$ is Kolmogorov-Loveland stochastic. Again taking $\overrightarrow{\beta}$ to converge to $\frac{1}{2}$ slowly enough that $\sum_{i=0}^{\infty}(\beta_i - \frac{1}{2})^2 = \infty$, this allowed Shen′ to conclude that not every Kolmogorov-Loveland stochastic sequence is random, thereby solving a twenty-year-old problem of Kolmogorov [7, 9] and Loveland [13, 14]. Theorem 4.14 has the following consequence concerning such sequences $\overrightarrow{\beta}$.

Corollary 4.15. *If $\overrightarrow{\beta}$ is a computable sequence of rational biases that converges to $\frac{1}{2}$ slowly enough that $\sum_{i=0}^{\infty}(\beta_i - \frac{1}{2})^2 = \infty$, then*

$$\mathrm{RAND}_{\overrightarrow{\beta}} \subseteq \mathrm{DIM}^1 - \mathrm{RAND}.$$

That is, every sequence that is random with respect to such a bias sequence $\overrightarrow{\beta}$ is an example of a sequence that has constructive dimension 1 but is not random.

Acknowledgment

I thank Elvira Mayordomo for very helpful discussions.

References

[1] J. Cai and J. Hartmanis. On Hausdorff and topological dimensions of the Kolmogorov complexity of the real line. *Journal of Computer and Systems Sciences*, 49:605–619, 1994.

[2] G. J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the Association for Computing Machinery*, 13:547–569, 1966.

[3] G. J. Chaitin. On the length of programs for computing finite binary sequences: statistical considerations. *Journal of the ACM*, 16:145–159, 1969.

[4] G. J. Chaitin. A theory of program size formally identical to information theory. *Journal of the Association for Computing Machinery*, 22:329–340, 1975.

[5] K. Falconer. *The Geometry of Fractal Sets.* Cambridge University Press, 1985.

[6] F. Hausdorff. Dimension und äusseres Mass. *Math. Ann.*, 79:157–179, 1919.

[7] A. N. Kolmogorov. On tables of random numbers. *Sankhyā, Series A*, 25:369–376, 1963.

[8] A. N. Kolmogorov. Three approaches to the quantitative definition of 'information'. *Problems of Information Transmission*, 1:1–7, 1965.

[9] A. N. Kolmogorov. Combinatorial foundations of information theory and calculus of probabilities. *Russian Mathematical Surveys*, 38:29–40, 1983.

[10] G. Kreisel. Note on arithmetical models for consistent formulae of the predicate calculus. *Fundamenta Mathematicae*, 37:265–285, 1950.

[11] L. A. Levin. On the notion of a random sequence. *Soviet Mathematics Doklady*, 14:1413–1416, 1973.

[12] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications.* Springer-Verlag, 1997.

[13] D. W. Loveland. The Kleene hierarchy classification of recursively random sequences. *Transactions of the American Mathematical Society*, 125:497–510, 1966.

[14] D. W. Loveland. A new interpretation of von Mises' concept of a random sequence. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 12:279–294, 1966.

[15] J. H. Lutz. Dimension in complexity classes. In *Proceedings of the Fifteenth Annual IEEE Conference on Computational Complexity*. IEEE Computer Society Press, 2000.

[16] P. Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619, 1966.

[17] P. Martin-Löf. Complexity oscillations in infinite binary sequences. *Zeitschrift für Wahrscheinlichkeitstheory und Verwandte Gebiete*, 19:225–230, 1971.

[18] P. Odifreddi. *Classical Recursion Theory*. Elsevier, 1989.

[19] B. Ya. Ryabko. Noiseless coding of combinatorial sources. *Problems of Information Transmission*, 22:170–179, 1986.

[20] B. Ya. Ryabko. Algorithmic approach to the prediction problem. *Problems of Information Transmission*, 29:186–193, 1993.

[21] B. Ya. Ryabko. The complexity and effectiveness of prediction problems. *Journal of Complexity*, 10:281–295, 1994.

[22] C. P. Schnorr. A unified approach to the definition of random sequences. *Mathematical Systems Theory*, 5:246–258, 1971.

[23] C. P. Schnorr. Zufälligkeit und Wahrscheinlichkeit. *Lecture Notes in Mathematics*, 218, 1971.

[24] C. P. Schnorr. Process complexity and effective random tests. *Journal of Computer and System Sciences*, 7:376–388, 1973.

[25] A. Kh. Shen′. The frequency approach to the definition of a random sequence. *Semiotika i Informatika*, 18:14–42, 1982. (In Russian.).

[26] A. Kh. Shen′. On relations between different algorithmic definitions of randomness. *Soviet Mathematics Doklady*, 38:316–319, 1989.

[27] R. J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7:1–22, 224–254, 1964.

[28] R. M. Solovay, 1975. reported in [**?**].

[29] L. Staiger. Kolmogorov complexity and Hausdorff dimension. *Information and Computation*, 102:159–194, 1993.

[30] L. Staiger. A tight upper bound on Kolmogorov complexity and uniformly optimal prediction. *Theory of Computing Systems*, 31:215–229, 1998.

[31] S. A. Terwijn. Personal communication, 2000.

[32] M. van Lambalgen. *Random Sequences*. PhD thesis, Department of Mathematics, University of Amsterdam, 1987.

[33] V. G. Vovk. On a randomness criterion. *Soviet Mathematics Doklady*, 35:656–660, 1987.

The Global Power of Additional Queries to p-Random Oracles

Wolfgang Merkle

Universität Heidelberg, Mathematisches Institut
Im Neuenheimer Feld 294, D-69120 Heidelberg, Germany
merkle@math.uni-heidelberg.de

Abstract. We consider separations of reducibilities by random sets. First, we show a result on polynomial-time bounded reducibilities which query their oracle non-adaptively: for every p-random set R, there is a set which is reducible to R with $k+1$ queries, but is not reducible to any other p-random set with at most k queries. This result solves an open problem stated in a recent survey paper by Lutz and Mayordomo [17]. Second, we show that the separation result above can be transferred from the setting of polynomial time bounds to a setting of rec-random sets and recursive reducibilities. This extends the main result of Book, Lutz, and Martin [8], who, by using different methods, showed a similar separation with respect to Martin-Löf-random sets. Moreover, in both settings we obtain similar separation results for truth-table versus bounded truth-table reducibility.

1 Introduction and Related Work

We consider separations of reducibilities in the context of resource-bounded measure theory. In the following, we use the symbol $\leq$ with appropriate sub- or superscripts to denote reducibilities, i.e., binary relations on Cantor space, the class of all sets of natural numbers. We say two reducibilities $\leq_r$ and $\leq_s$ are separated by an oracle A if the lower spans of A with respect to these reducibilities, i.e., the classes $\{X : X \leq_r A\}$ and $\{X : X \leq_s A\}$, differ. It is easy to see that two reducibilities are different (as binary relations on Cantor space) if and only if they are separated by some oracle. Beyond this simple observation, the question of which reducibilities are separated by what kind of oracles has been the object of intensive studies. Here, for a given pair of reducibilities, typical question are the following. Are there separating oracles of low complexity? How comprising is the class of separating oracles? Which properties are sufficient for being a separating oracle?

Ladner, Lynch, and Selman [13] considered separations of the usual polynomial-time bounded reducibilities in the range between p-m- and p-T-reducibility (see Sect. 2 for definitions). They showed that for every distinct pair of such reducibilities, there is a separating oracle which can be computed in exponential time. Subsequently, in their seminal paper [6], Bennett and Gill obtained results about separations by almost all oracles, i.e., they showed that for certain

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 914–926, 2000.
© Springer-Verlag Berlin Heidelberg 2000

pairs of reducibilities the class of separating oracles has measure 1 with respect to uniform measure on Cantor space. In fact, for every $k > 0$, every pair of distinct reducibilities chosen among p-T-, p-tt, p-btt, p-btt$(k+1)$, and p-btt(k)-reducibility can be separated by random oracles, see [16,21], as well as [10] for a separation of p-btt$(k+1)$- and p-btt(k)-reducibility by almost all tally oracles.

A separation by almost all oracles can be expressed equivalently by saying that the class of oracles which do not separate the reducibilities under consideration has uniform measure 0. Lutz and Mayordomo [16] could show for certain pairs of polynomial-time bounded reducibilities of truth-table type that the class of non-separating oracles does not just have uniform measure 0 but is in fact covered by a polynomial-time computable martingale. In particular, they showed that for every natural number k, there is a polynomial-time computable martingale which covers all oracles which do not separate p-btt$(k+1)$- and p-btt(k)-reducibility, whence, in particular, these reducibilities are separated by every p-random oracle. The latter can be rephrased by saying that these reducibilities are locally separated by the class of p-random oracles. Here, formally, a nonempty class $\mathbf{C}$ locally separates two given reducibilities iff for every set A in $\mathbf{C}$, the lower spans of A with respect to these reducibilities are different.

We say a class $\mathbf{C}$ globally separates two given reducibilities in case for every set A in $\mathbf{C}$ there is a set B which is reducible to A with respect to one of the given reducibilities but B is not reducible to any set in $\mathbf{C}$ with respect to the other reducibility. Moreover, in case such a set B exists not for all but just for some sets A in $\mathbf{C}$, we say that $\mathbf{C}$ yields a weak global separation of the reducibilities under consideration. The definition of global separation is symmetric in the reducibilities involved, however, for reducibilities $\leq_r$ and $\leq_s$ where $X \leq_r Y$ implies $X \leq_s Y$, sets A and B as above must satisfy $B \leq_s A$ and $B \not\leq_r A$ (in fact, $B \not\leq_r Z$ for all Z in $\mathbf{C}$), and similar remarks hold for the other concepts of separation mentioned so far. In distinguishing local and global separation we follow Book, Lutz, and Martin [8], who discuss such separations for the classes of Martin-Löf-random, tally, and sparse sets.

In the sequel we will consider global separations by various classes of random sets. Such investigations can be viewed as part of a more comprising research project where one asks which types of reductions are able to transform random objects into what types of far from random objects – see [12] and [18] for results in this direction and for further discussion and references.

Remark 1. By definition, every local or global separation by a class $\mathbf{C}$ extends trivially to every nonempty subclass of $\mathbf{C}$.

In Theorem 4, we show that the class of p-random oracles yields a global separation of p-btt$(k+1)$- and p-btt(k)-reducibility. This, together with Remark 6, solves Problem 7 in the recent survey article [17], where it has been asked to prove or disprove that, in our terms, the class of p-random oracles yields a weak global separation of these reducibilities. In Sect. 5, then we obtain by basically the same proof as for Theorem 4 that for every natural number k, the class of rec-random sets globally separates p-btt$(k+1)$-reducibility from btt(k)-reducibility, i.e., from the reducibility restricted to at most k non-adaptive queries where the

reductions are computed by total Turing machines which might run in arbitrary time and space. By Remark 1, this yields as a special case the main result of Book, Lutz, and Martin [8], who showed, by using different methods, a corresponding global separation with respect to the class of Martin-Löf-random sets, which is a proper subclass of the class of rec-random sets. Moreover, we will argue that in both settings, i.e., for polynomial-time bounded, as well as for recursive reductions and martingales, the corresponding random sets globally separate the corresponding notions of truth-table and bounded truth-table reducibility.

2 Notation

The notation used in the following is mostly standard, for unexplained notation refer to [4], [7], and [15]. All strings are over the alphabet $\Sigma = \{0, 1\}$. We identify strings with natural numbers via the isomorphism which takes the length-lexicographical ordering on $\{\lambda, 0, 1, 00, \dots\}$ to the usual ordering on ω, the set of natural numbers. If not explicitly stated differently, the terms set and class refer to sets of natural numbers and to sets of sets of natural numbers, respectively.

A partial characteristic function is a (total) function from some subset of the natural numbers to $\{0, 1\}$. A partial characteristic function is finite iff its domain is finite. The restriction of a partial characteristic function β to a set I is denoted by $\beta|I$, whence in particular for a set X, the partial characteristic function $X|I$ has domain I and agrees there with X. We identify strings of length n in the natural way with a partial characteristic function with domain $\{0, \dots, n-1\}$, whence in particular strings can be viewed as prefixes of sets. For a partial characteristic function α with domain $\{z_0 < \dots < z_{n-1}\}$, the string associated with α is the (unique) string β where $\beta(j) = \alpha(z_j)$ for $j = 0, \dots, n-1$. For a set X and a partial characteristic function σ we write $\langle X, \sigma \rangle$ for the set which agrees with σ for all arguments in the domain of σ and which agrees with X, otherwise.

We will consider the following polynomial-time bounded reducibilities: Turing reducibility (p-T), truth-table reducibility (p-tt), where the queries have to be asked non-adaptively, bounded truth-table reducibility (p-btt), where for each reduction the number of queries is bounded by a constant, and, even more restrictive, p-btt(k)-reducibility, where for all reductions this constant is bounded by the natural number k. The relation symbol $\leq^{\mathrm{p}}_{\mathrm{btt}}$ refers to p-btt-reducibility, and relation symbols for other reducibilities are defined in a similar fashion. Expressions such as p-T-reduction and $\leq^{\mathrm{p}}_{\mathrm{T}}$-reduction will be used interchangeably. We represent p-btt-reductions by a pair of polynomial time computable functions g and h where $g(x)$ gives the set of strings queried on input x and $h(x)$ is a truth-table of a Boolean function over k variables which specifies how the answers to the queries in the set $g(x)$ are evaluated. Here we assume, firstly, via introducing dummy variables, that the cardinality of $g(x)$ is always exactly k and, secondly, by convention, that for $i = 1, \dots, k$, the ith argument of the Boolean function $h(x)$ is assigned the ith query in $g(x)$.

3 Resource-Bounded Measure

We give a brief introduction to resource-bounded measure which focusses on the concepts that will be used in subsequent sections. For more comprehensive accounts of resource-bounded measure theory see the recent survey papers by Ambos-Spies and Mayordomo [4] and by Lutz [15].

The theory of resource-bounded measure is usually developed in terms of martingales, which can be viewed as payoff functions of gambles of the following type. A player successively places bets on the individual bits of the characteristic sequence of an unknown set A or, for short, the player bets on A. The betting proceeds in rounds $i = 1, 2, \ldots$ where during round i, the player receives the length $i - 1$ prefix of A and then, firstly, decides whether to bet on the ith bit being 0 or 1 and, secondly, determines the stake by specifying the fraction of the current capital which shall be bet. Formally, a player can be identified with a betting strategy $b : \{0,1\}^* \to [-1, 1]$ where the bet is placed on the next bit being 0 or 1 depending on whether $b(w)$ is negative or nonnegative, respectively, and where the absolute value of the real $b(w)$ is the fraction of the current capital that shall be at stake.

The player starts with strictly positive, finite capital. At the end of each round, in case the current guess has been correct, the capital is increased by this round's stake and, otherwise, is decreased by the same amount. So given a betting strategy b, we can inductively compute the corresponding payoff function d by applying the equations

$$d(w0) = d(w) - b(w) \cdot d(w) \qquad\qquad d(w1) = d(w) + b(w) \cdot d(w) \ .$$

Intuitively speaking, the payoff $d(w)$ is the capital the player accumulates till the end of round $|w|$ by betting on a set which has the string w as a prefix. Conversely, every function d from strings to nonnegative reals which for all strings w, satisfies the fairness condition

$$d(w) \ = \ \frac{d(w0) + d(w1)}{2} \ , \tag{1}$$

induces canonically a betting function b, where

$$b(w) \ = \ \frac{d(w1) - d(w0)}{2} \cdot \frac{1}{d(w)}$$

in case $d(w)$ differs from 0 and $b(w) = 0$, otherwise. We call a function d from strings to nonnegative reals a martingale iff $d(\lambda) > 0$ and d satisfies the fairness condition (1) for all strings w.

By the preceding discussion it follows for gambles as described above that the possible payoff functions are exactly the martingales and that in fact there is a one-to-one correspondence between martingales and betting strategies. We will frequently identify martingales and betting strategies via this correspondence and, if appropriate, notation introduced for martingales will be extended to the induced betting strategies.

We say a martingale d succeeds on a set A if d is unbounded on the prefixes of A, i.e., if $\limsup_{n \text{ in } \omega} d(A|\{0,\dots,n\}) = \infty$, and d succeeds on or covers a class iff d succeeds on every set in the class. It has been shown by Ville that a class has uniform measure 0 iff the class can be covered by some martingale (see [26,4]). Thus every countable class and, in particular, most of the classes considered in complexity and recursion theory can be covered by martingales, whence in order to distinguish such classes in terms of coverability one has to restrict the class of admissible martingales. In the context of recursion theory, this led to the consideration of recursive martingales, whereas in connection with complexity classes one has to impose additional resource-bounds, see [23,24,25,27] and [1,4,14,15,20], respectively.[1] Here, in general, for a given class $\boldsymbol{C}$ one is interested in finding a class of martingales which allows the covering of interesting subclasses of $\boldsymbol{C}$, but not of $\boldsymbol{C}$ itself.

In connection with measure on complexity classes, most attention has been received by measure concepts for the exponentially time-bounded classes $\mathbf{E} = \mathbf{DTIME}(2^{\text{lin}})$ and $\mathbf{EXP} = \mathbf{DTIME}(2^{\text{poly}})$. For example, in the case of the class $\mathbf{E}$, Lutz proposed to use martingales which on input w are computable in time polynomial in the length of w.

We say a set is p-random if the set cannot be covered by a polynomial-time computable martingale, and we write p-RAND for the class of all p-random sets. The notion rec-random set and the class rec-RAND of all rec-random sets are defined likewise with recursive martingales in place of polynomial-time computable ones. Moreover, we will consider Martin-Löf-random sets, which have been introduced in [19] and have been characterized equivalently by Schnorr [24] as the sets which cannot be covered by so-called subcomputable martingales. The classes of p-random, rec-random, and Martin-Löf random sets all have uniform measure 1, whence each of these classes of random sets can, in a sense, be viewed as class of typical sets. For a proof it suffices to observe that the class of sets on which a single martingale succeeds always has uniform measure 0 and, by σ-additivity, the same holds for every countable union of such classes.

We conclude this section by two remarks in which we describe standard techniques for the construction of martingales.

Remark 2. Let a finite set D be given, as well as a list $\langle D_1,\dots,D_m\rangle$ of pairwise disjoint subsets of D which all have the same cardinality $k > 0$. Then for a partial characteristic function σ with domain D and a string w of length k we might ask for the frequency

$$\alpha(\sigma, w, \langle D_1,\dots,D_m\rangle) \quad := \quad \frac{|\{j : w \text{ is the associated string of } \sigma|D_j\}|}{m}$$

with which w occurs in σ as associated string at the positions specified by the D_i. In case the sets D_i are clear from the context, we suppress mentioning them and write $\alpha(\sigma, w)$, for short.

[1] An effective martingale d is always confined to rational values and is computed by a Turing machine which on input w outputs an appropriate finite representation of $d(w)$.

If we choose the bits of σ by independent tosses of a fair coin, then for every w of length k, the expected value of $\alpha(\sigma, w)$ is $1/2^k$. It is suggestive to assume that for large m, only for a small fraction of all partial characteristic functions with domain D the frequency of w will deviate significantly from the expected value. Using Chernoff bounds (see for example Lemma 11.9 in [22]), one can indeed show that given k and a rational $\varepsilon > 0$, we can compute a natural number $m(k, \varepsilon)$ such that for all $m \geq m(k, \varepsilon)$ and for all D and $D_1, \ldots, D_m$ as above we have

$$\frac{|\{\sigma : D \to \{0,1\} : (\frac{1}{2} \cdot \frac{1}{2^k} < \alpha(\sigma, w, \langle D_1, \ldots, D_m \rangle) < \frac{3}{2} \cdot \frac{1}{2^k}\}|}{2^{|D|}} \quad \geq \quad 1 - \varepsilon \ . \tag{2}$$

Remark 3. Let I be a finite set and let Θ be a subset of all partial characteristic functions with domain I. We can easily construct a martingale which by betting on places in I, increases its capital by a factor of $2^{|I|}/|\Theta|$ for all sets B where $B|I$ is in Θ. Here the martingale takes the capital available when betting on the minimal element of I and distributes it evenly among the elements of Θ, then computing values upwards according to the fairness condition for martingales.

4 Separations by p-Random Oracles

Theorem 4 extends Lutz and Mayordomo's local separation of p-btt$(k+1)$- and p-btt(k)-reducibility in [16] to a global separation. Recall that the lower $\leq$-span of a class $\boldsymbol{C}$ is the class of all sets which are $\leq$-reducible to some set in $\boldsymbol{C}$.

Theorem 4. *Let R be a p-random set and let k be a natural number. Then the lower p-btt$(k+1)$-span of R is not contained in the lower p-btt(k)-span of* p-RAND.

Proof. In order to define a set A and a p-btt$(k+1)$-reduction (g_0, h_0) from A to R we let $h_0(x)$ be the truth-table of the $(k+1)$-place conjunction and we let

$$g_0(x) \ := \ \{x0^1 1^{k+1}, x0^2 1^k, \ldots, x0^{k+1} 1^1\} \ , \qquad A \ := \ \{x : g_0(x) \subseteq R\} \ . \tag{3}$$

We are done if we can show that if A is p-btt(k)-reducible to a set, then this set cannot be p-random. So let B be an arbitrary set and assume that A is reducible to B via the p-btt(k)-reduction (g, h). We will construct a polynomial-time computable martingale d which succeeds on B. To this end, we define a sequence $n_0, n_1, \ldots$ with

$$n_0 = 0 \ , \qquad n_{i+1} > 2^{n_i} \ , \qquad \log n_{i+1} > m\left(k+1, \frac{1}{2^{i+1}}\right) \tag{4}$$

(here $m(.,.)$ is the function defined in Remark 2) and such that given x of length n, we can compute in time $\mathcal{O}(n^2)$ the maximal i with $n_i \leq n$. Such a sequence

can be obtained by standard methods, for details refer to the chapter on uniform diagonalization and gap languages in [7].

It is helpful to view the betting strategy of the martingale d as being performed in stages $i = 0, 1, \ldots$ where the bets of stage i depend on the g-images of the strings of length n_i. While considering the queries made for strings of length n_i with $i > 0$, we will distinguish short queries with length strictly less than

$$l_i \quad := \quad \left\lfloor \frac{n_i}{2k} \right\rfloor \tag{5}$$

and long queries, i.e. queries of length at least l_i. We call two strings x and y equivalent iff, for some i, both have identical length n_i and in addition we have

$$(i)\ h(x) = h(y) \ , \qquad (ii)\ \{z \text{ in } g(x) : |z| < l_i\} = \{z \text{ in } g(y) : |z| < l_i\} \ , \tag{6}$$

i.e., two strings of length n_i are equivalent iff they have the same truth-table and the same set of short queries. Then for some constant c, the number of equivalence classes of strings of length n_i can be bounded as follows

$$2^{2^k} \sum_{j=0}^{k} \binom{2^{l_i} - 1}{j} \quad \leq \quad 2^{2^k}(k+1) \cdot 2^{l_i \cdot k} \quad \leq \quad c \cdot 2^{\frac{n_i}{2k} \cdot k} \quad = \quad c \cdot 2^{\frac{n_i}{2}} \ .$$

So the 2^{n_i} strings of length n_i are partitioned into at most $c \cdot 2^{\frac{n_i}{2}}$ equivalence classes, whence for all sufficiently large i, there is an equivalence class of cardinality at least $m_i := \lfloor \log n_i \rfloor$. For all such i, among all equivalence classes of strings of length n_i we choose one with maximal cardinality (breaking ties by some easily computable but otherwise arbitrary rule), we let J_i contain the first m_i strings in this equivalence class, and we let

$$\alpha_i \quad = \quad \frac{|A \cap J_i|}{|J_i|} \ . \tag{7}$$

Claim 1. For almost all i,

$$\frac{1}{2} \cdot \frac{1}{2^{k+1}} \quad < \quad \alpha_i \quad < \quad \frac{3}{2} \cdot \frac{1}{2^{k+1}} \ . \tag{8}$$

Proof. Fix an index i and assume that (8) is false. Let $z_1 < \ldots < z_{m_i}$ be the elements of J_i, let $D_j = g_0(z_j)$ for $j = \{1, \ldots, m_i\}$, and let D be the union of D_1 through D_{m_i}. If we let $w = 1^{k+1}$, then by definition of h_0 we have $\alpha_i = \alpha(R|D, w)$, whence (8) remains false with α_i replaced by $\alpha(R|D, w)$. On the other hand, (8) is false with α_i replaced by $\alpha(\sigma, w)$ for at most a $1/2^i$-fraction of all partial characteristic functions σ with domain D because by (4) and the choice of the m_i, we have $m_i \geq m(k+1, 1/2^i)$. Remark 3 then shows that while betting on R, a martingale can increase its capital by a factor of 2^i by betting for all places in D on the $1/2^i$-fraction of partial characteristic functions for which (8) is false. Based on the latter fact we now construct a martingale, where we

leave it to the reader to show that the martingale can be computed in polynomial time. The initial capital 1 is split into $c_1, c_2, \ldots$ where $c_i = 1/2^i$ is exclusively used to place bets on the strings in the set D which corresponds to the index i, i.e., the strings which are in $g_0(x)$ for some x in J_i. By the preceding discussion, the martingale can increase the capital c_i to at least 1 for all i such that (8) is false. But if this were the case for infinitely many values of i, the martingale would succeed on R, thus contradicting the assumption that R is p-random. □

While constructing the martingale d which is meant to succeed on B, we will exploit that by Claim 1 for almost all i, the density of the set A on J_i is confined to a small interval around the comparatively low value $1/2^{k+1}$. Let Γ be the functional which corresponds to the btt(k)-reduction given by (g, h) – whence for example A is equal to $\Gamma(B)$ – and for all $i \geq i_0$, let

$$H_i \quad = \quad \bigcup_{x \text{ in } J_i} \{z : z \text{ in } g(x) \text{ and } |z| \geq l_i\} \ ,$$

i.e., H_i is the set of all long queries made by strings in J_i. Then we can argue that only for a fraction of all partial characteristic functions σ with domain H_i the set $\Gamma(\langle B, \sigma\rangle)$ has such low density on J_i. Formally, for every $i > i_0$ and for every partial characteristic function σ with domain H_i, we let

$$\beta_i(\sigma) \ = \ \frac{|\Gamma(\langle B, \sigma\rangle) \cap J_i|}{|J_i|} \ , \qquad \rho \ = \ \frac{3}{2} \cdot \frac{1}{2^{k+1}} \ ,$$

and, further,

$$\Theta_i \ = \ \{\sigma : \sigma \text{ partial characteristic function with domain } H_i \text{ and } \beta_i(\sigma) < \rho\} \ .$$

By Claim 1 for almost all i, the restriction of B to H_i must be contained in Θ_i. We will argue next that there is some $\delta < 1$ such that for almost all i, the set Θ_i comprises at most a δ-fraction of all partial characteristic functions with domain H_i. We will then exploit the latter fact in the construction of the martingale d by betting against the $(1 - \delta)$-fraction of partial characteristic functions outside of Θ_i which have already been ruled out as possible restriction of B to H_i.

For the moment, let τ_x be the Boolean function obtained from $h(x)$ by hard-wiring $B(z)$ into $h(x)$ for all short queries z in $g(x)$. Recall that by convention queries are assigned to variables in length-lexicographical order, whence for equivalent strings x and y, the Boolean functions τ_x and τ_y are identical. Thus for every i, all strings in J_i are mapped to the same Boolean function, which we denote by τ_i. We call a Boolean function constant iff it evaluates to the same truth value for all assignments to its arguments (whence in particular all 0-placed Boolean functions are constant).

Claim 2. For almost all i, τ_i is not constant.

Proof. If τ_i is constant, then the value $A(x)$ must be the same for all x in J_i. But then α_i is either 0 or 1, whence Claim 1 implies that this is the case for at most finitely many indices i. □

Claim 3. There is a constant $\delta < 1$ such that for almost all i, the set Θ_i comprises at most a δ-fraction of all partial characteristic functions with domain H_i.

Proof. For given i such that τ_i is not constant, consider the random experiment where we use independent tosses of a fair coin in order to choose the individual bits of a random partial characteristic function $\widehat{\sigma}$ with domain H_i. Then all partial characteristic functions of the latter type occur with the same probability, whence the fraction we want to bound is just the probability of picking an element in Θ_i.

For every string x in J_i, define a 0-1-valued random variable b_x and, moreover, define a random variable γ_i with rational values in the closed interval $[0,1]$ by

$$b_x(\widehat{\sigma}) := \Gamma(\langle B, \widehat{\sigma}\rangle, x) \ , \qquad \gamma_i(\widehat{\sigma}) := \frac{1}{|J_i|} \sum_{x \text{ in } J_i} b_x(\widehat{\sigma}) \ .$$

Consider an arbitrary string x in J_i. By assumption, τ_i is not constant, whence there is at least one assignment to $\widehat{\sigma}$ such that b_x is 1. Moreover such an assignment occurs with probability at least $1/2^k$ because $h(x)$, and thus also τ_i, has at most k variables. Thus the expected value of b_x is at least $1/2^k$ and by linearity of expectation we obtain

$$E(\gamma_i) \quad = \quad \frac{1}{|J_i|} \sum_{x \text{ in } J_i} E(b_x) \quad \geq \quad \frac{1}{|J_i|} \sum_{x \text{ in } J_i} \frac{1}{2^k} \quad = \quad \frac{1}{2^k} \ . \tag{9}$$

If we let p be the probability of the event "$\gamma_i < \rho$", we have

$$\frac{1}{2^k} \leq E(\gamma_i) \leq p \cdot \rho + (1-p) \cdot 1 \leq \rho + (1-p) = \frac{3}{4} \cdot \frac{1}{2^k} + (1-p) \ , \tag{10}$$

where the relations follow, from left to right, by (9), by definition of p and by $\gamma_i \leq 1$, because the probability p is bounded by 1, and by definition of ρ. But (10) is obviously false in case $(1-p)$ is strictly less than $1/2^{k+2}$, whence p can be bounded from above by $\delta := 1 - 1/2^{k+2}$. □

For all i, let $I_i = \{x : l_i \leq |x| < l_{i+1}\}$. The n_i grow sufficiently fast such that for some i_1 and for all $i > i_1$, the set H_i is contained in I_i. Moreover, by Claim 3, for some i_2 and all $i > i_2$, there is a set Θ_i of partial characteristic functions with domain H_i where, firstly, Θ_i contains only a δ-fraction of all such partial characteristic functions, and, secondly, Θ_i contains the restriction of B to H_i. Let i_3 be the maximum of i_1 and i_2.

Now we are in a position to describe a betting strategy which succeeds on B. On input w, let x be the $(|w|+1)$th string, i.e., the string on which we might bet. We first compute the index i such that x is in I_i, together with the corresponding set H_i. In case $i \leq i_3$ or if x is not in H_i, we abstain from betting. Otherwise, we place a bet on x according to a betting strategy as described in Remark 3, which, while placing bets on the strings in H_i, increases the capital by a factor of at least $1/\delta$ by betting against the partial characteristic functions which are not in Θ_i. Here all necessary computations can be performed in time $2^{\mathcal{O}(n_i)}$ and

hence, by $|x| \geq l_i = \lfloor n_i/2k \rfloor$, in time $2^{\mathcal{O}(|x|)}$. It follows that this betting strategy induces a polynomial-time computable martingale which on interval I_i preserves its capital in case $i \leq i_3$ and increases its capital by a factor of at least $1/\delta$ for all $i > i_3$. This finishes the proof of Theorem 4. □

Remark 5. The assertion of Theorem 4 remains valid if we simply require the set R to be n-random instead of p-random, (i.e., if we require that there is no martingale computable in time $\mathcal{O}(n)$ which succeeds on R). For a proof, note that Ambos-Spies, Terwijn, and Zheng have shown in [5] that for every n^2-random set R, there is a p-random set R_0 which is p-m-reducible to R while, in fact, the latter assertion is true for n-random R. Now the relaxed version of Theorem 4 follows because the existence of a separating set A as required in the theorem extends directly from R_0 to R.

Remark 6. Theorem 4 states that the lower p-btt(k+1)-span of every p-random set R contains a set A which is not in the lower p-btt(k)-span of any p-random set. As already noted in [8], for a set R which is not just p-random but is even Martin-Löf-random, such a set A cannot be recursive. This follows from the fact that every recursive sets which is p-btt($k+1$)-reducible to a Martin-Löf-random set is in fact computable in polynomial time. The latter fact is attributed to folklore in [18] and can be obtained as a special case of a result of Book, Lutz, and Wagner in [9].

Corollary 7. *For every p-random set R, the lower p-tt-span of R is not contained in the lower p-btt span of* p-RAND.

Proof. For a given p-random set R and for every k, let A_{k+1} be defined in the same way as the set A has been defined in (3) in the proof of Theorem 4, whence A_{k+1} is p-btt(k+1)-reducible to R, but is not p-btt(k)-reducible to any p-random set. If we let

$$B \quad = \quad \{x : x = 1^k 0y \text{ and } y \text{ in } A_k\}$$

then by the definition of the sets A_k the set B is p-tt-reducible to R. On the other hand, if B were p-btt-reducible to some p-random set R_0, then B were in fact p-btt(k)-reducible to R_0 for some k. But then, in particular, A_{k+1} were p-btt(k)-reducible to R_0, thus contradicting the choice of A_{k+1}. □

5 Separations by Rec-random Oracles

Lutz [14] showed that recursive martingales yield a reasonable measure concept for the class of recursive sets, where in particular the class of all recursive sets cannot be covered by a recursive martingale (see [24,25,27] for discussion of measure concepts for the class of recursive sets).

Next we state two results on rec-random sets which correspond rather closely to Theorem 4 and Corollary 7 on p-random sets. In connection with Theorem 8

and Corollary 9, recall from the introduction that rec-RAND is the class of sets which cannot be covered by a recursive martingale. Moreover, let btt-reducibility be defined like p-btt-reducibility, except that a btt-reduction is required to be computed by a total Turing machine which might run in arbitrary time and space, and let btt(k)-reducibility be the restriction of btt-reducibility where the number of queries is bounded by k.

Theorem 8. *Let the set R be in* rec-RAND *and let k be a natural number. Then the lower* p-$(k+1)$-tt-*span of R is not contained in the lower btt(k)-span of* rec-RAND*.*

Corollary 9. *For every set R in* rec-RAND*, the lower p-tt-span of R is not contained in the lower btt-span of* rec-RAND*.*

Due to space considerations, we omit the proofs of Theorem 8 and Corollary 9, which are essentially the same as in the case of p-random sets.

Remark 10. Recall from Remark 1 that a global separation by a class $\boldsymbol{C}$ extends to all nonempty subclasses of $\boldsymbol{C}$. Furthermore, note that Schnorr [24] has implicitly shown that the class of Martin-Löf-random sets is a proper subclass of rec-RAND. As a consequence, the global separation by the class rec-RAND stated in Theorem 8 yields as a corollary the main result of Book, Lutz, and Martin in [8] who showed that for all k, the lower p-btt($k+1$)-span of a Martin-Löf-random set is never contained in the lower btt(k)-span of the class of all Martin-Löf-random sets.

Acknowledgements

We are grateful to Elvira Mayordomo for several useful corrections and suggestions, where the latter include a significant simplification of the argument in Remark 5 via referring to [5]. Furthermore, we would like to thank Klaus Ambos-Spies, Jack Lutz, and Jan Reimann for helpful discussion. Finally, we are grateful for the comments of an anonymous referee of ICALP 2000.

References

1. E. Allender and M. Strauss. Measure on small complexity classes, with applications for BPP. In: *Proceedings of the 35th Annual IEEE Symposium an Foundations of Computer Science*, p. 807–818, IEEE Computer Society Press, 1994.
2. K. Ambos-Spies. Randomness, relativizations, and polynomial reducibilities. In *Proc. First Structure in Complexity Theory Conference*, Lecture Notes in Computer Science 223, pages 23–34. Springer-Verlag, 1986.
3. K. Ambos-Spies, E. Mayordomo, Y. Wang, and X. Zheng. Resource-bounded balanced genericity, stochasticity and weak randomness. In C. Puech, R. Reischuk, editors, 13th Annual Symposium on Theoretical Aspects of Computer Science 1996, Lecture Notes in Computer Science 1046, p. 63–74, Springer-Verlag, 1996.

4. K. Ambos-Spies and E. Mayordomo. Resource-bounded measure and randomness. In: A. Sorbi, editor, *Complexity, logic, and recursion theory*, p. 1-47. Dekker, New York, 1997.
5. K. Ambos-Spies, S. A. Terwijn, and X. Zheng. Resource bounded randomness and weakly complete problems. *Theoretical Computer Science*, 172:195–207, 1997.
6. Ch. H. Bennett and J. Gill. Relative to a random oracle A, $P^A \neq NP^A \neq co{-}NP^A$ with probability 1. *SIAM Journal on Computing*, 10:96–113, 1981.
7. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity*, volume I and II. Springer-Verlag, 1995 and 1990.
8. R. V. Book, J. H. Lutz, and D. M. Martin Jr. The global power of additional queries to random oracles. *Information and Computation*, 120:49-54, 1995. A preliminary version appeared in [11], p. 403–414.
9. R. V. Book, J. H. Lutz, and K. W. Wagner. An observation on probability versus randomness with applications to complexity classes. *Mathematical Systems Theory* 27:201–209, 1994.
10. R. V. Book and S. Tang. Polynomial-time reducibilities and "almost all" oracle sets. *Theoretical Computer Science* 81:35–47, 1991.
11. P. Enjalbert, E. W. Mayr, and K. W. Wagner, editors, *11th Annual on Symposium on Theoretical Aspects of Computer Science 1994*, Lecture Notes in Computer Science 775, Springer-Verlag, 1994.
12. D. W. Juedes, J. I. Lathrop, and J. H. Lutz. Computable depth and reducibility. *Theoretical Computer Science*, 132:37–70, 1994.
13. R. E. Ladner, N. A. Lynch, and A. L. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science* 1:103–123, 1975.
14. J. H. Lutz. Almost everywhere high nonuniform complexity. *Journal of Computer and System Sciences*, 44:220–258, 1992.
15. J. H. Lutz. The quantitative structure of exponential time. In: Hemaspaandra, Lane A. et al., editors, *Complexity theory retrospective* II, p. 225–260, Springer-Verlag, 1997.
16. J. H. Lutz and E. Mayordomo. Cook versus Karp-Levin: separating completeness notions if NP is not small. *Theoretical Computer Science* 164:141–163, 1996. A preliminary version appeared in [11], p. 415–426.
17. J. H. Lutz and E. Mayordomo. Twelve problems in resource-bounded measure. *Bulletin of the European Association for Theoretical Computer Science* 68:64–80, 1999.
18. J. H. Lutz and D. L. Schweizer. Feasible reductions to Kolmogorov-Loveland stochastic sequences. *Theoretical Computer Science* 225:185-194, 1999.
19. P. Martin-Löf. The definition of random sequences. *Information and Control* 9:602–619, 1966.
20. E. Mayordomo. *Contributions to the Study of Resource-Bounded Measure.* Doctoral dissertation, Universitat Politècnica de Catalunya, Barcelona, Spain, 1994.
21. W. Merkle and Y. Wang. Random separations and "Almost" classes for generalized reducibilities. In J. Wiedermann and P. Hájek, editors, *Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 969, p. 179–190. Springer-Verlag, 1995. Submitted for further publication.
22. C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.
23. C.-P. Schnorr. A unified approach to the definition of random sequences. *Mathematical Systems Theory*, 5:246–258, 1971 .
24. C.-P. Schnorr. *Zufälligkeit und Wahrscheinlichkeit.* Lecture Notes in Mathematics 218, Springer-Verlag, 1971.

25. S. A. Terwijn. *Computability and Measure*. Doctoral dissertation, Universiteit van Amsterdam, Amsterdam, Netherlands, 1998.
26. J. Ville. *Étude Critique de la Notion de Collectif*. Gauthiers-Villars, 1939.
27. Y. Wang. *Randomness and Complexity*. Doctoral dissertation, Universität Heidelberg, Mathematische Fakultät, INF 288, Heidelberg, Germany, 1996.

Homogenization and the Polynomial Calculus

Josh Buresh-Oppenheim[⋆,1], Toniann Pitassi[⋆⋆,1],
Matt Clegg[2], and Russell Impagliazzo[⋆⋆⋆,2]

[1] Computer Science Department, University of Arizona
Tucson, AZ 85721, USA
Fax: 1-520-621-4246
{bureshop, toni}@cs.arizona.edu
[2] Computer Science and Engineering, University of California, San Diego
La Jolla, CA, USA
{mclegg, russell}@cs.ucsd.edu

Abstract. In standard implementations of the Gröbner basis algorithm, the original polynomials are homogenized so that each term in a given polynomial has the same degree. In this paper, we study the effect of homogenization on the proof complexity of refutations of polynomials derived from Boolean formulas in both the Polynomial Calculus (PC) and Nullstellensatz systems. We show that the PC refutations of homogenized formulas give crucial information about the complexity of the original formulas. The minimum PC refutation degree of homogenized formulas is equal to the Nullstellensatz refutation degree of the original formulas, whereas the size of the homogenized PC refutation is equal to the size of the PC refutation for the originals. Using this relationship, we prove nearly linear ($\Omega(n/logn)$ vs. $O(1)$) separations between Nullstellensatz and PC degree, for a family of explicitly constructed contradictory 3CNF formulas. Previously, a $\Omega(n^{1/2})$ separation had been proved for equations that did not correspond to any CNF formulas, and a $\log n$ separation for equations derived from kCNF formulas.

1 Introduction

Buchberger's algorithm is a very popular technique from algebraic geometry, which is used to find a Gröbner basis for a family of polynomial equations over variables $x_1, \ldots, x_n$.

Buchberger's algorithm can be applied to solve SAT. Starting with an initial boolean formula in conjunctive normal form, $C = C_1 \wedge C_2 \wedge \ldots \wedge C_m$, convert each clause C_i into an equivalent polynomial equation (over some field F), and add the additional equations $x_i^2 - x_i = 0$ to force 0/1 solutions. The corresponding family of polynomial equations will have a solution over F if and only if C is satisfiable. Conversely, C is

⋆ Research partially supported by NSF grant CCR-9457782 and a scholarship from the Arizona Chapter of the ARCS Foundation.

⋆⋆ Research supported by NSF grant CCR-9457782 and US-Israel BSF Grant 95-00238.

⋆⋆⋆ Research supported by NSF CCR-9734911, Sloan Research Fellowship BR-3311, and by a cooperative research grant INT-9600919/ME-103 from NSF and the MŠMT (Czech Republic), and USA-Israel-BSF Grant 97-00188

U. Montanari et al. (Eds.): ICALP 2000, LNCS 1853, pp. 926–938, 2000.
© Springer-Verlag Berlin Heidelberg 2000

unsatisfiable if and only if 1 is in the ideal generated by these polynomials, and hence is in the Gröbner basis.

Buchberger's algorithm has many unspecified aspects, such as a term order, and the order in which S-remainders are computed. Any specification of these parameters yields a valid Gröbner basis algorithm, but the running time can vary highly depending on these issues. Many heuristics and modifications have been suggested and implemented to make the algorithm simpler or faster. However, typically the algorithm is applied in the context of infinite fields, and thus the heuristics commonly considered may be meaningless or counter-productive in our setting. We are interested in understanding which heuristics work well and why in our setting. One basic heuristic is homogenization. The original system of polynomials is replaced by an equivalent system that is homogeneous, i.e., all terms of a polynomial in the system have the same degree. For systems of polynomials derived from Boolean tautologies, we show that homogenization basically creates a hybrid between two well-studied proof systems, Nullstellensatz (HN) and Polynomial Calculus (PC).

The Nullstellensatz (HN) and Polynomial Calculus (PC) proof systems, first defined in [BIK$^+$96, CEI96], are algebraic proof systems for refuting systems of unsolvable polynomial equations. They have been extensively studied in the past several years, due to their connections to standard proof systems, [Pit97, BIK$^+$97] and NP-search classes, as well as Buchberger's algorithm. The two common complexity measures for proofs in these systems are degree and size. We show that the *size* of PC proofs is preserved under homogenization. However, the *degree* can increase dramatically. In fact, the degree of PC proofs for the homogenized polynomials is exactly that of HN proofs for the original polynomials. Using this characterization, we are able to derive an almost optimal separation for PC and HN degrees. We give explicit 3-CNF contradictions whose translations have $O(1)$ degree PC proofs, but require $\Omega(n/\log n)$ degree HN proofs. Previously, a $\Omega(n^{1/2})$ separation had been proved for a system of Boolean polynomials that did not correspond to any CNF ([CEI96, Bus97]), and a $\log n$ separation for equations derived from $kCNF$ formulas [BP96].

It follows, from the first result, that if our term order uses only the degree of the homogenizing variable as a tie-breaker, homogenization is guaranteed not to substantially change the time of Buchberger's algorithm for Satisfiability. However, the second result indicates this might not be the case for degree-respecting term orders, as are used in standard implementations.

2 Background

2.1 Gröbner Bases

The theory of Gröbner bases requires an ordering on the terms of the polynomial ring in which we operate. In this case, we choose an arbitrary ordering on the variables and use any induced order on the terms (such as lex, grlex, etc). We use the following definition and theorem, which are both standard to the theory of Gröbner bases ($deg(f)$ is the degree of f, LT is the largest term under the ordering, and LCM is the least common multiple):

Definition 1. *A finite subset G of an ideal I (over a polynomial ring R) is called a Gröbner basis if it generates I, and if the set $\{\mathrm{LT}(g) \mid g \in G\}$ generates the monomial ideal $\mathrm{LT}(I) = \{\mathrm{LT}(f) \mid f \in I\}$.*

Theorem 1. *[CO92] For G a basis for I and $g_1, g_2 \in G$, let*

$$S(g, g') = \frac{\mathrm{LCM}(\mathrm{LT}(g), \mathrm{LT}(g'))}{\mathrm{LT}(g)} g - \frac{\mathrm{LCM}(\mathrm{LT}(g), \mathrm{LT}(g'))}{\mathrm{LT}(g')} g'.$$

G is a Gröbner basis for I if and only if for all $g, g' \in G$, there exist $\{a_f\}_{f \in G} \subset R$, such that

$$S(g, g') = \sum_{f \in G} a_f f$$

and $\deg(a_f f) \leq \deg(S(g, g'))$ for all $f \in G$.

$S(g, g')$ is called the S-polynomial of g and g'. The S-remainder, written $S(g, g') \bmod G$, is the remainder of $S(g, g')$ divided by (the elements of) G (listed in some fixed order). Informally the above theorem states that if G is a basis for I, then G is Gröbner if and only if, for all $g, g' \in G$, the S-remainder of g and g' is zero. This theorem gives rise to the following algorithm, commonly called Buchberger's algorithm, for constructing a Gröbner basis. The input to the algorithm is a set of polynomials $F = (f_1, \ldots, f_s)$. Initially, the basis (called G) contains F. At each step, we select a pair of polynomials in G, compute their S-remainder and, if it is non-zero, we add it to G. The algorithm terminates when all pairs of polynomials in G have S-remainders of zero.

This algorithm is a cornerstone of computational algebraic geometry. Many heuristics have been invented and analyzed for improving the runtime. However, in most applications, the algorithm is applied to infinite fields, and thus it is not clear whether these heuristics make sense in our setting, where the underlying field is finite and solutions are 0/1 valued.

We also mention a well-known lemma for computing S-remainders:

Lemma 1. *[CO92] Let g and g' be two polynomials such that*

$$\gcd(\mathrm{LT}(g), \mathrm{LT}(g')) = 1.$$

The S-remainder of g and g' is 0.

2.2 Homogeneous Gröbner Bases

Let F be a finite set of polynomials. Let I_F be the ideal generated by F, and let $I_F(d)$ be the subset of the ideal consisting of all those polynomials in I_F of degree at most d. For solving instances of SAT, we are interested in knowing whether or not $I_F(0)$ is empty.

It is natural to consider a truncated version of the Gröbner basis algorithm where we ignore all S-polynomials of degree greater than d. We will let $[d] - Grobner(F)$ denote the output of this truncated version of the algorithm applied to F. It would be nice if

$[d] - Grobner(F)$ had all of the nice properties of a Gröbner basis for those polynomials in $I_F(d)$. In particular, we would like the largest terms of $[d] - Grobner(F)$ to generate the largest terms of $I_F(d)$. However, in general this is not the case since S-remainders of high degree can lead to new basis elements of very low degree.

On the other hand, the truncated Gröbner basis algorithm does have this nice property when applied to *homogeneous* polynomials. For our purposes, a polynomial P is *homogeneous* if every term in P has the same degree. If every polynomial in F is homogeneous, it can easily be seen that all non-zero S-polynomials will also be homogeneous, and all polynomials output by the Gröbner basis algorithm will be homogeneous. Moreover, to test a particular polynomial f for membership in I_F, it suffices to compute $[d] - Grobner(F)$, where $deg(f) = d$.

Because of this and other nice properties, common implementations of the Gröbner basis algorithm begin by homogenizing F, if it is not already homogeneous. To do this, a new variable Z is introduced, that is last in the variable ordering. Before running the algorithm, each initial equation $f_i \in F$ is modified (by multiplying each term by a power of Z) so that each term in f_i has degree equal to $deg(f_i)$.

The trade-off ensues from the fact that, in the homogenized setting, the polynomials in the ideal may have higher degree than their corresponding polynomials in the non-homogenized setting (i.e. there could be extra factors of Z increasing their degree. We will see that, while a non-homogenized PC-proof consists of testing for elements in $I_F(0)$, we must check for membership of Z^c, for some *a priori* unknown constant c, to prove the homogenized case.) In this paper we analyze the complexity of the homogenized versus non-homogenized approach, applied to equations derived from 3CNF formulas.

2.3 Algebraic Proof Systems

In this paper, we consider two particular algebraic proof systems (i.e. systems under which clauses of an unsatisfiable logical formula are translated into algebraic equations which are then proven to be contradictory). The first is the Hilbert Nullstellensatz (HN) system and the second, the Polynomial Calculus (PC) system. Both rely on the fact that given a contradictory set of polynomials, $Q_1, \ldots, Q_m \in K[X]$ for some field K, those polynomials generate the unit ideal in the ring $K[X]$. In other words, the equations do not have a solution in the algebraic closure of K if and only if 1 is in the ideal generated by $Q_i(\overline{x})$. There are several ways of characterizing the elements of this ideal in terms of linear combinations of the generators. Such a demonstration that 1 is in the ideal is thus a *proof* of the unsolvability of the equations Q_i. The Nullstellensatz and Polynomial Calculus systems are based on two such characterizations. The standard versions of both assume the variables are Boolean, that is, they take $x^2 - x$ as axiomatic. However, the homogenizing variable will not be Boolean, so we need to consider the extensions of these systems to non-Boolean systems.

Under HN, a proof or refutation is given by exhibiting a sum, $\sum_{i=1}^{m} P_i Q_i = 1$, for any $\{P_i\}_{i=1}^{m} \subset K[X]$. The degree of this derivation, then, is $\max\{\deg(P_i Q_i) \mid 1 \leq i \leq m\}$. Its size is $\sum_{i=1}^{m} \mathrm{size}(P_i)$ where $\mathrm{size}(P_i)$ is the number of monomials in the polynomial P_i. The HN degree of a set of contradictory polynomials is the degree of the minimum-degree HN proof.

A PC derivation of $Q \in K[X]$ from $Q_1, \ldots, Q_n \in K[X]$ is a sequence of polynomials $P_1, \ldots, P_m = Q$, where each P_i is either

1. Q_j for some j.
2. mP_j for $j < i$ and m a term in $K[X]$.
3. $aP_j + bP_l$ for $j, l < i$ and $a, b \in K$.

The size of this derivation is l. Its degree is $\max\{\deg(P_i) \mid 1 \leq i \leq l\}$. The PC degree of Q from a set of polynomials is the degree of the minimum-degree PC derivation of Q from those polynomials. If no such derivation exists (i.e. $Q \notin \langle Q_1, \ldots, Q_m \rangle$), then the PC degree of Q is ∞. A PC proof or refutation of a set of contradictory polynomials is a PC derivation of 1 from those polynomials. A PC refutation of a set of contradictory polynomials homogenized by Z is a PC derivation of Z^c for any integer $c \geq 0$. The PC degree of a set of contradictory, non-homogenized polynomials is the degree of the minimum-degree proof of those polynomials. The PC degree of a set of contradictory, homogenized polynomials is the minimum over all c of the PC degree of Z^c from those polynomials. Notice that, since a PC proof allows cancellation of terms at each step, its degree is always at most the HN-degree for the same set of polynomials.

3 Relationships between Complexity Measures

The following theorem shows that the homogenized PC degree and the HN-degree are basically the same.

Theorem 2. *Let* $\{q_1, \ldots, q_m\} \subset K[x_1, \ldots, x_n]$. *Let* $\{Q_1, \ldots, Q_m\} \subset K[X_1, \ldots, X_n, Z]$ *be the homogenizations of the above polynomials. Then,* $Z^k \in \langle Q_1, \ldots, Q_m \rangle$ *iff* $\{q_1, \ldots, q_m\}$ *has a degree* k *Hilbert Nullstellensatz (HN) refutation.*

Proof. First assume $\{q_1, \ldots, q_m\}$ has a degree k HN refutation: $f = \Sigma p_i q_i = 1$, for some $\{p_1, \ldots, p_m\} \subset K[x_1, \ldots, x_n]$ such that $\max \deg(p_i q_i) = k$. Let f_α be the terms of f with multidegree $\alpha = (d_1, \ldots, d_n)$ and $\Sigma d_i = d$. Clearly $f_\alpha = 0$ for all non-trivial α. Now let $F = \Sigma Z^{k - \deg(pq)} P_i Q_i$, where P_i is the homogenization of p_i. Now, the terms of f_α have become the terms of multidegree $\alpha' = (d_1, \ldots, d_n, k - d)$. Thus, for non-trivial α, $F_{\alpha'} = 0$. For $\alpha = 0$, $f_\alpha = 1$, so $F_{\alpha'} = Z^k$. Hence, $F = Z^k$.

Now assume $Z^k \in \langle Q_1, \ldots, Q_m \rangle$. Then we have $\Sigma p_i Q_i = Z^k$ for some $\{p_1, \ldots, p_m\} \subset K[X_1, \ldots, X_n, Z]$. But we can remove all the terms from p_i such that $\deg(p_i Q_i) \geq k$ since they must all cancel. Let's say this yields p'_i. Then we set $Z = 1$ on both sides of the equation, so we have $\Sigma(p'_i|_{Z=1}) q_i = 1$ where each term in the sum has degree k. Hence $\{q_1, \ldots, q_m\}$ has a degree k HN refutation.

Corollary 1. *The degree of the Polynomial Calculus (PC) refutation for* $\{Q_1, \ldots, Q_m\}$ *is equal to the degree of the HN refutation for* $\{q_1, \ldots, q_m\}$.

Proof. A PC refutation for $\{Q_1, \ldots, Q_m\}$ consists of deriving Z^k for some k. Clearly the degree of this derivation must be at least k. But by the theorem, there is a degree k HN refutation of $\{q_1, \ldots, q_m\}$, so $\deg_{PC}(Q_1, \ldots, Q_m) \geq \deg_{HN}(q_1, \ldots, q_m)$. On the other hand, if there is a degree k HN refutation of $\{q_1, \ldots, q_m\}$, then $\{Q_1, \ldots, Q_m\}$ derive Z^k and we saw in the above proof that this can be done in degree k, so $\deg_{PC}(Q_1, \ldots, Q_m) \leq \deg_{HN}(q_1, \ldots, q_m)$.

The following theorem, proven in [BIK+97] shows that the degree of tree-like PC refutations is very close to the degree of HN refutations.

Theorem 3. *If there is a degree d HN refutation of Q, then there is a tree-like, degree d PC refutation of Q. Conversely, if there is a tree-like, degree d, size S PC refutation of Q, then there is a HN refutation of Q of degree $O(d \log S)$.*

Lastly, we show that the size of a homogenized PC refutation is no larger than the size of a PC refutation.

Theorem 4. *Let $q = \{q_1, \ldots, q_m\}$ be a family of polynomial equations and let $Q = \{Q_1, \ldots, Q_m\}$ be the homogenizations of the above polynomials, with Z being the new variable introduced by homogenization. Then there exists an i such that if q has a size s PC refutation, then QZ^i has a size $O(s)$ PC refutation.*

Proof. Let P be a PC refutation of q. We will show by induction on the number of lines in P, $|P|$, that for all polynomials r, if there exists a size s proof of r from q, then there exists an i and a size $O(s)$ proof of RZ^i from Q, where R is the homogenization of r. For the base case, suppose that r has a one line proof from q. Then r is an equation of q, so R is an initial equation of Q and we are done. Assume the inductive hypothesis holds for l lines, and now let r have an $l+1$-line proof from q. There are two cases to consider. The first case is where $r = xt$, where t has an l-line proof from q. In this case, by induction there exists i such that TZ^i has a small proof from Q. But then $R = TZ^i x$ is a proof from Q and we are done. The other case is when $r = r_1 + r_2$, where r_1 and r_2 each have short proofs from q. Without loss of generality, suppose that $deg(r_1) = deg(r_2) + c$. Then $R = R_1 + R_2 Z^c$. By induction, there exist i, j such that there are small proofs of $R_1 Z^i$, and $R_2 Z^j$ from Q. If $j = c + i$, we have $(R_1 - R_2 Z^c)Z^i$ and we are done. Otherwise, if $j > c + i$, multiply $R_1 Z^i$ by Z^δ where $\delta = j - c - i$. Then we have $(R_1 - R_2 Z^c)Z^{i+\delta}$. Finally, if $j < c + i$, let $\delta = c + i - j$ and multiply $R_2 Z^j$ by Z^δ and get $(R_1 - R_2 Z^c)Z^i$.

4 Lower Bound for HN-Degree

4.1 Tautologies on Graphs

The idea behind the tautologies first appeared in [RM97], and has also appeared in subsequent papers [BEGJ98, BSW99]. We begin with a directed, acyclic graph, D, where each node has constant indegree. This graph is viewed as a general implication graph as follows: (1) there is one variable, $X(v)$, corresponding to each vertex v in the graph; (2) for each vertex v in S, the set of sources, the corresponding variable $X(v)$ is true; (3) for each non-source vertex v with parent vertices $P(v)$, we have that if all of the variables corresponding to vertices in $P(v)$ are true, then $X(v)$ is also true; (4) finally, to make the tautology false, for some sink vertex t, $X(t)$ is false.

Throughout this section, we will assume there is only one sink (if not, identify all the sinks to one node). Also, we will abuse notation and use v for $X(v)$. The meaning should be clear from the context. Formally, we define the following clauses: for any $v \notin S$, we have the implication $\bigwedge_{u \in P(v)} u \to v$. Also, we have the axioms s for each

$s \in S$. Finally, we insist on $\bar{t}$. If the indegree of D is k, then the above formula is a $k+1$-CNF formula.

For algebraic proof systems, we fix a field K and convert the above implications into equations in the ring $K[X(V)]$ (we use the notation $\mathrm{prod}(U)$ to mean $\prod_{u \in U} X(u)$ for U a set of nodes):

$$v\ \mathrm{prod}(P(v)) - \ \mathrm{prod}(P(v)) = 0.$$

We also include equations restricting the variables to boolean values: $v^2 - v = 0$. Again, if the indegree of D is k, then these equations have degree $k+1$.

The natural way to refute the above formula/equations is to begin at the source vertices, and derive successively that each layer of vertices must be true, until finally we can conclude that each sink vertex must be true. This gives us the desired contradiction since there is some sink vertex that is false. For any graph D with indegree k, this natural refutation can be formalized as a PC refutation of degree $k+1$, and also as a polynomial-size tree-like Resolution refutation. However, we show here that if the graph is sufficiently complicated (it has high pebbling number), then any HN refutation must have high degree.

Our lower bound strategy will be to take advantage of Corollary 1. That is, we will implicitly describe a Gröbner basis for the above equations, and then prove that the degree of this basis is at least as large as the pebbling number associated with the underlying graph.

4.2 Gröbner Basis for Graph Tautologies

Consider the homogenized implications and restrictions (for each v):

$$v\ \mathrm{prod}(P(v)) - Z\ \mathrm{prod}(P(v)) = 0, \qquad v^2 - vZ = 0.$$

Here Z is the variable added for homogenization. We also have the assertions (for each source s):

$$s - Z = 0,$$

which will be considered implications saying that the empty set implies s, and (for one sink t):

$$t = 0.$$

Let I be the ideal generated by these equations. In practise, the proof of our lower bound is focused on implications. Hence, instead of deriving Z^d (for some d), we simply try to derive the equation

$$f = (t\ \mathrm{prod}(S) - Z\ \mathrm{prod}(S))Z^d.$$

To see that this works and is still a proof, assume that we have derived Z^d. Then clearly we can derive f (or any other multiple of Z^d). On the other hand, if we have derived f then

$$-f + (Z - t)(\sum_{i=1}^{|S|}(s_i - Z) \prod_{j=i+1}^{|S|} s_j) + t = Z^{d+|S|+1}.$$

Although we have derived a higher power of Z than we assumed, the proof remains valid. As for the degree of the proof, we use the concept of *Z-degree*:

Definition 2. *Let $P \in K[X, Z]$ be a homogenized polynomial. We define the Z-degree of P, $\mathrm{zdeg}(P)$ to be the maximal $m \in \mathbf{N}$ such that $P/Z^m \in K[X, Z]$.*

The Z-degree of f, then, is d and we shall consider this the *de facto* degree of the proof. Notice that a lower bound for $\mathrm{zdeg}(f)$ is a lower bound for the power of Z which we can derive and hence is a lower bound on the HN-degree. Now we are ready to explore the Gröbner basis for our ideal. We first exhibit certain polynomials in the ideal:

Proposition 1. *Let $V_1, V_2 \subset V$, $u_1, u_2 \in V$, $d_1, d_2 \geq 0$ and assume that the implications*

$$f_1 = (u_1 \operatorname{prod}(V_1) - Z \operatorname{prod}(V_1))Z^{d_1}, \qquad f_2 = (u_2 \operatorname{prod}(V_2) - Z \operatorname{prod}(V_2))Z^{d_2}$$

are in I. If $u_1 \in V_2$, then let $U = V_1 \cup V_2 - \{u_1\}$ and let $d = \max\{d_1, d_2\}$. We can then conclude that $f_3 = (u_2 \operatorname{prod}(U) - Z \operatorname{prod}(U))Z^{d+1}$ is in I.

Proof. $(Z^{d-d_2} \operatorname{prod}(V_1 - V_2))f_2 - (u_2 - Z)(Z^{d-d_1} \operatorname{prod}(V_2 - V_1 - \{u_1\}))f_1 = (u_2 \operatorname{prod}(U) - Z \operatorname{prod}(U))Z^{d+1} = f_3$.

Let $V \to u$ denote the formula stating that if all of the variables in V are true, then u is also true. Informally the above proposition says that if we have a degree d_1 PC derivation of $V_1 \to u_1$, and a degree d_2 PC derivation of $V_2 \to u_2$, where $u_1 \in V_2$, then we can derive $V_1 \cup V_2 - \{u_1\} \to u_2$ in degree $max(d_1, d_2) + 1$ (notice that this is still true for the degenerate case where $V_1 = \emptyset$). Let G be the set of equations formed recursively as follows:

1. G contains all of the generators of I.
2. For all implications f_1 and f_2 (see above) in G, f_3 is in G.

G, then, is just the closure under applications of Proposition 1. It is not hard to see that, invariably, the conclusion node of each implication is not included among the hypothesis nodes, since the hypothesis nodes are all predecessors of the conclusion node and the graph is acyclic. We shall use this fact in proving that G is a very fundamental set:

Theorem 5. *G is a Gröbner basis for I.*

Proof. Clearly G is a basis since it contains all of the generators of I. We show that it is Gröbner using Theorem 1 (recall that we defined Z to be last in the ordering of the variables). Consider two implications,

$$f_1 = (u_1 \operatorname{prod}(V_1) - Z \operatorname{prod}(V_1))Z^{d_1},$$

$$f_2 = (u_2 \operatorname{prod}(V_2) - Z \operatorname{prod}(V_2))Z^{d_2}.$$

The S-remainder is

$$S(f_1, f_2) = (\operatorname{prod}(V_1 \cup V_2 \cup \{u_2\} - \{u_1\})Z - \operatorname{prod}(V_2 \cup V_1 \cup \{u_1\} - \{u_2\})Z)Z^d.$$

Here and throughout, we omit any cases of two polynomials with relatively prime largest terms by Lemma 1. Hence there are three remaining possibilities for implications:

1. $u_1 = u_2$: If $u_1 = u_2$, then the S-remainder becomes

$$(\operatorname{prod}(V_1 \cup V_2)Z - \operatorname{prod}(V_1 \cup V_2)Z)Z^d.$$

But the two terms are clearly equal, so the S-remainder is 0.

2. $u_1 \in V_2$: Recall from above that $u_2 \notin V_1$ since V_1 consists of predecessors of u_1, u_1 is a predecessor of u_2 and the graph is acyclic. Consider the first term of the S-remainder:

$$t_1 = \operatorname{prod}(V_1 \cup V_2 \cup \{u_2\} - \{u_1\})Z^{d+1}.$$

But we know that

$$g = (u_2 \operatorname{prod}(V_1 \cup V_2 - \{u_1\}) - Z \operatorname{prod}(V_1 \cup V_2 - \{u_1\}))Z^{d+1}$$

is in G by definition. So t_1 can be reduced to a lower-degree term:

$$t_1 - g = \operatorname{prod}(V_1 \cup V_2 - \{u_1\})Z^{d+2}.$$

The second term is $t_2 = \operatorname{prod}(V_2 \cup V_1 \cup \{u_1\} - \{u_2\})Z^{d+1}$. Since $u_2 \notin V_1$, this is the same as $\operatorname{prod}(V_2 \cup V_1 \cup \{u_1\})Z^{d+1}$. Also, we have $f_1 \in G$, so we can reduce the second term:

$$t_2 - \operatorname{prod}(V_2 - (V_1 \cup \{u_1\}))Z^{d-d_1+1} f_1 = \operatorname{prod}(V_1 \cup V_2 - \{u_1\})Z^{d+2}.$$

These two expressions are the same, so we have reduced $S(f_1, f_2)$ to 0.

3. $u_1 \neq u_2$, $u_1 \notin V_2$ and $u_2 \notin V_1$, but $V_1 \cap V_2 \neq \emptyset$: Now, the S-remainder is

$$(\operatorname{prod}(V_1 \cup V_2 \cup \{u_2\})Z - \operatorname{prod}(V_1 \cup V_2 \cup \{u_1\})Z)Z^d.$$

Let t_1 be the first term and t_2 be the second. Then,

$$t_1 - \operatorname{prod}(V_1 - V_2)Z^{d+1-d_2} f_2 = \operatorname{prod}(V_1 \cup V_2)Z^{d+2}.$$

Similarly,

$$t_2 - \operatorname{prod}(V_2 - V_1)Z^{d+1-d_1} f_1 = \operatorname{prod}(V_1 \cup V_2)Z^{d+2}.$$

Again, we have reduced the S-remainder to 0.

This concludes the S-remainders for every pair of implications in G. We now consider the S-remainders involving the boolean restrictions. Let $h = u^2 - uZ$. Then,

$$S(h_1, f_2) = \operatorname{prod}(V_2 \cup \{u_2\} - \{u\})uZ^{d_2+1} - u \operatorname{prod}(\{u\} - \{u_2\} - V_2) \operatorname{prod}(V_2)Z^{d_2+1}.$$

Again we have cases:

1. $u = u_2$: We can then assume that $u \notin V_2$, so the S-remainder becomes

$$u \operatorname{prod}(V_2)Z^{d_2+1} - u \operatorname{prod}(V_2)Z^{d_2+1} = 0.$$

2. $u \in V_2$ and $u \neq u_2$: In this case, the S-remainder is

$$\mathrm{prod}(V_2 \cup \{u_2\})Z^{d_2+1} - u\ \mathrm{prod}(V_2)Z^{d_2+1}.$$

If we call the first term t_1 and the second t_2, consider $t_1 - Zf_2 = \mathrm{prod}(V_2)Z^{d_2+2}$. We can rewrite t_2 as $u^2\ \mathrm{prod}(V_2 - \{u\})Z^{d_2+1}$. But then $t_2 - \mathrm{prod}(V_2 - \{u\})$ $Z^{d_2+1}h_1 = u\ \mathrm{prod}(V_2 - \{u\})Z^{d_2+2}$. But this is just $\mathrm{prod}(V_2)Z^{d_2+2}$ so we are done.

Finally, we have the odd case of the equation $t = 0$.

1. Consider h_1, where $u = t$. $S(h_1, t) = tZ$, but t is in G, so it reduces to 0.
2. Similarly, we have f_1, where $u_1 = t$. $S(f_1, t) = tZ^{d_1+1}$, so again it reduces to 0.

We have now shown that all the S-remainders are 0 modulo G and can conclude (by Theorem 1) that G is Gröbner .

4.3 Optimal Lower Bounds and Pebbling

Strong lower bounds for specific graphs will easily follow by showing that any HN derivation can easily be converted into an efficient pebbling strategy for the corresponding graph. Interesting connections between pebbling and propositional proofs were made previously in [ET99, BSW99].

Definition 3. *Let $D = (V, E)$ be a directed, acyclic graph. A configuration is a subset of V. A legal pebbling of a vertex v in D is a sequence of configurations, the first being the empty set and the last being $\{v\}$ and in which each configuration C' follows from the previous configuration C by one of the following rules:*

1. *v can be added to C to get C' if all immediate predecessors of v are in C.*
2. *Any vertex can be removed from C to obtain C'.*

The complexity of a legal pebbling of v is the maximal size of any configuration in the sequence. The pebbling number of a graph D with a single sink vertex s is the minimal number n such that there exists a legal pebbling of s with complexity n.

Lemma 2. *Let D be a directed, acyclic graph, and let Q_D be the corresponding unsatisfiable system of homogeneous equations corresponding to the implication formula associated with D. If Q_D has a degree d PC refutation, then D has pebbling complexity $O(d)$.*

Proof. In Theorem 5, we gave a recursive characterization of the polynomials in the Grobner basis. We'll show by induction on the depth of a polynomial in this recursion, that if $(u\ \mathrm{prod}(U) - Z\ \mathrm{prod}(U))Z^d$ is in the Groebner basis, then u can be pebbled from U with $d+k$ pebbles. The base case is the axioms $v\ \mathrm{prod}(P(v)) - Z\ \mathrm{prod}(P(v))$, which can always be pebbled with k pebbles (if k is the maximum in-degree of D). Otherwise, $(u\ \mathrm{prod}(U) - Z\ \mathrm{prod}(U))Z^d$, was derived from $(v\ \mathrm{prod}(V_1) - Z\ \mathrm{prod}(V_1))Z_1^d$, and $(u\ \mathrm{prod}(V_2) - Z\ \mathrm{prod}(V_2))Z_2^d$, where $d = \max d_1, d_2 + 1$, $v \in V_2$, $v \notin V_1$ and

$U = V_1 \cup V_2 - \{v\}$; these formulas having been already derived. Thus, by the induction assumption, we can pebble v from V_1 with $d_1 + k$ pebbles and u from V_2 with $d_2 + k$ pebbles. Then we can pebble u from U as follows. First, pebble v from $V_1 \subseteq U$. Then, leaving a pebble on v, pebble u from $V_2 \subseteq U \cup \{v\}$. The number of pebbles is at most the larger of $d_1 + k$ and $d_2 + k + 1$, which is at most $d + k$.

We note that the above lemma is not tight, as can be seen with the linear graph corresponding to the induction principle. In this case, the pebbling number is 2, whereas the degree is $\log n$. We do, however, get the following result:

Theorem 6. *There is a directed acyclic graph with constant in-degree that requires HN degree $\Omega(n/logn)$.*

Proof. [CPT77] exhibits a graph based on a construction by Valiant that has n nodes and in-degree 2, but has pebbling measure $\Omega(n/logn)$. By the lemma, we are done.

5 Comparison with Resolution

The PC system gives rise to Buchberger's algorithm to solve SAT as mentioned above. The HN system gives rise to a simpler algorithm for SAT whereby one solves a system of *linear* equations. That is, if we start with a system of equations Q (including $x^2 - x = 0$ for all variables x), and we assume they have a degree d HN proof, then it can be found as follows. Consider an alleged degree d proof, $\sum_i P_i Q_i = 1$, where the polynomials P_i are indeterminants, represented by vectors of variables, x_i^t, where x_i^t represents the coefficient in front of term t in the polynomial P_i. Solving for the P_i's amounts to solving a sparse system of linear equations (in the variables x_i^t), where we have one equation for each term of degree at most d. This algorithm has runtime $n^{O(d)}$, and hence is efficient if d is small ([CEI96] gives similar upper bounds for this algorithm and Buchberger's algorithm, although in the case of Buchberger's algorithm, the parameter d might be smaller).

Complete algorithms for SAT with the best empirical performance are Resolution based, and typically are variations on the Davis-Loveland-Logeman (DLL) procedure. Here we show that with respect to worst-case complexity, the algorithms are incomparable.

Lemma 3. *The graph tautologies mentioned above (with maximum pebbling number) have polynomial-size Tree-like Resolution (DLL) proofs but require nearly linear degree HN proofs.*

Proof. To see that the graph tautologies always have small Tree-like Resolution proofs, simply order the vertices of the graph in such a way that if v is a vertex with parents $P(v)$, then all vertices in $P(v)$ appear before v in the ordering. Then a tree-like Resolution proof can be constructed as a decision tree, where we query the vertices according to the ordering described above. The height of the tree will be equal to the number of vertices, but the width will be constant.

Lemma 4. *The Tseitin graph principles (mod p) have constant degree HN proofs in the field GF_p but require exponential-size Resolution proofs.*

Proof. [Urq87] has shown that the Tseitin graph principles require exponential-size Resolution proofs. To see that they have polynomial-size HN proofs, consider the clauses that correspond to a particular vertex, expressing that the number of edges incident to that vertex is odd. Each such clause converts into an initial equation, and the set of equations assocated with a vertex v can be used to derive the equation stating that the sum of the vertices incident to that vertex is 1 mod 2. Since the total number of variables mentioned in these equations is constant, the HN degree of this derivation is constant. Do this for every vertex, and then simply add up the resulting equations to derive $1 = 0$.

Acknowledgments

We would like to thank Will Evans for very helpful conversations.

References

[BEGJ98] M.L. Bonet, J. L. Esteban, N. Galesi, and J. Johannsen. Exponential separations between restricted resolution and cutting planes proof systems. In *Proceedings from 38th FOCS*, pages 638–647. 1998.

[BIK+96] Paul W. Beame, Russell Impagliazzo, Jan Krajíček, Toniann Pitassi, and Pavel Pudlák. Lower bounds on Hilbert's Nullstellensatz and propositional proofs. *Proc. London Math. Soc.*, 73(3):1–26, 1996.

[BIK+97] S. Buss, R. Impagliazzo, J. Krajíček, P. Pudlák, A. A. Razborov, and J. Sgall. Proof complexity in algebraic systems and bounded depth Frege systems with modular counting. *Computation Complexity*, 6(3):256–298, 1997.

[BP96] S. R. Buss and T. Pitassi. Good degree bounds on Nullstellensatz refutations of the induction principle. In *Proceedings of the Eleventh Annual Conference on Computational Complexity (formerly: Structure in Complexity Theory)*, pages 233–242, Philadelphia, PA, May 1996. IEEE.

[BSW99] E. Ben Sasson and A. Wigderson. Short proofs are narrow–resolution made simple. In *Proceedings of 31st ACM STOC*, pages 517–526. 1999.

[Bus97] S. R. Buss. Lower bounds on Nullstellensatz proofs via designs. In P. W. Beame and S. R. Buss, editors, *Proof Complexity and Feasible Arithmetics*, DIMACS, pages 59–71. American Math. Soc, 1997.

[CEI96] M. Clegg, J. Edmonds, and R. Impagliazzo. Using the Gröbner basis algorithm to find proofs of unsatisfiability. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 174–183, Philadelphia, PA, May 1996.

[CO92] J. Cox, D. Little and D. O'Shea. *Ideals, Varieties and Algoirthms*. Springer-Verlag, 1992.

[CPT77] R. Celoni, W.J. Paul, and R.E. Tarjan. Space bounds for a game on graphs. *Math. Systems Theory*, 10:239–251, 1977.

[ET99] J. L. Esteban and Jacobo Toran. Space bounds for resolution. In *Proc. 16th STACS*, pages 551–561. Springer-Verlag LNCS, 1999.

[Pit97] T. Pitassi. Algebraic propositional proof systems. In *DIMACS Series in Discrete Mathematics*, volume 31, pages 215–243. American Math. Soc, 1997.

[RM97] R. Raz and P. McKenzie. Separation of the monotone nc hierarchy. In *Proceedings of 38th IEEE Foundations of Computer Science*. 1997.

[Urq87] A. Urquhart. Hard examples for resolution. *Journal of the ACM*, 34(1):209–219, 1987.

Author Index

Samson Abramsky 1
Micah Adler 756
William Aiello 463
Ali Akhavi 373
Noga Alon 576
Stephen Alstrup 73
Christoph Ambühl 305
Sunil Arya 624
Albert Atserias 151

Christel Baier 780
Ganesh Baliga 844
Cyril Banderier 388
Olivier Baudron 499
Birgit Baum-Waidner 524
Marie-Pierre Béal 561
Michael A. Bender 809
Sandeep Bhatt 463
Johannes Blömer 248
Beate Bollig 187
Endre Boros 588
Olivier Bournez 793
Mario Bravetti 744
Andrei Z. Broder 808
Roberto Bruni 175
Andrei A. Bulatov 272
Josh Buresh-Oppenheim 926

Christian Cachin 512
Cristiano Calcagno 25
Jan Camenisch 512
Olivier Carton 561
John Case 844
Bogdan S. Chlebus 717
Matt Clegg 926
Giovanni Di Crescenzo 451
János Csirik 296
Artur Czumaj 856

Evgeny Dantsin 236
Volker Diekert 211
Hristo N. Djidjev 821

Michael Elkin 636
Lars Engebretsen 2
Gregor Engels 127
Javier Esparza 475
José Espírito Santo 600

Faith Fich 756
Philippe Flajolet 388
Riccardo Focardi 354
W.J. Fokkink 729
Hervé Fournier 832

Bernd Gärtner 305
Nicola Galesi 151
Paul Gastin 211
Ricard Gavaldà 151
Leszek Gąsieniec 717
Dan R. Ghica 103
Andreas Goerdt 236
Leslie Ann Goldberg 705, 756
Oded Goldreich 687
Roberto Gorrieri 354, 744
Michelangelo Grigni 869
Vladimir Gurvich 588

Torben Hagerup 61
Johan Håstad 235
Boudewijn Haverkort 780
Reiko Heckel 127
Keijo Heljanko 475
Matthew Hennessy 415
Jesper G. Henriksen 675
Holger Hermanns 780
Edward A. Hirsch 236
Jacob Holm 73
Jonas Holmerin 2
Juraj Hromkovič 199

Russell Impagliazzo 926

Klaus Jansen 878
Peter Jeavons 272
Mark Jerrum 705

Sampath Kannan 705
Haim Kaplan 576
Juhani Karhumäki 199, 536
Richard M. Karp 428
Joost-Pieter Katoen 780
Leonid Khachiyan 588
Joe Kilian 512
Hartmut Klauck 199
Barbara König 403
Pascal Koiran 832
Michael Krivelevich 13, 576
Andrei A. Krokhin 272
Antonín Kučera 317
K. Narayan Kumar 675
V.S. Anil Kumar 624
Dietrich Kuske 648

Leonid Libkin 260
Andrzej Lingas 856
Gerald Lüttgen 163
Denis Lugiez 342
S.P. Luttik 729
Jack H. Lutz 902

Kazuhisa Makino 588
Oded Maler 793
Dahlia Malkhi 576
Zohar Manna 429
Luciano Margara 768
Keye Martin 116
Fabio Martinelli 354
Richard Mayr 329
Guy McCusker 103
Pierre McKenzie 890
B. Meenakshi 487
Kurt Mehlhorn 571
Michael Mendler 163
Wolfgang Merkle 844, 914
Eugenio Moggi 25
Ben C. Moszkowski 223
Joy Müller 512
Madhavan Mukund 675
Alan Mycroft 37

Frank Neven 547

Anna Östlin 717
Rafail Ostrovsky 463

Mike Paterson 705, 756
David Peleg 636
Giuseppe Persiano 451
Ion Petre 536
Seth Pettie 49
Reinhard Pichler 612
Toniann Pitassi 926
Gordon Plotkin 85
David Pointcheval 499
Lorant Porkolab 878
John Power 85

S. Raj Rajagopalan 463
Vijaya Ramachandran 49
R. Ramanujam 487
H. Ramesh 624
James Riely 415
John Michael Robson 717
Dana Ron 809

Donald Sannella 85
Alfredo De Santis 451
Vladimiro Sassone 175
Gilles Schaeffer 388
Georg Schnitger 199
Philippe Schnoebelen 342
Uwe Schöning 236
Thomas Schwentick 547, 890
Sebastian Seibert 199
Steven S. Seiden 283
Richard Sharp 37
Janos Simon 768
Henny B. Sipma 429
Michèle Soria 388
Bernhard von Stengel 305
Frank Stephan 844
Jacques Stern 499
Julien Stern 576

Walid Taha 25
Robert Tennent 85
Denis Thérien 890
P.S. Thiagarajan 675

Tomasz Fryderyk Urbański 663

Brigitte Vallée 373
Heribert Vollmer 890
Van H. Vu 13

Michael Waidner 524
Ingo Wegener 187
Gerhard J. Woeginger 296