

Lecture Notes in Computer Science

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

1893

Springer

Berlin

Heidelberg

New York

Barcelona

Hong Kong

London

Milan

Paris

Singapore

Tokyo

Mogens Nielsen Branislav Rován (Eds.)

Mathematical Foundations of Computer Science 2000

25th International Symposium, MFCS 2000
Bratislava, Slovakia, August 28 – September 1, 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

Mogens Nielsen
University of Aarhus, Department of Computer Science
Ny Munkegade, Bldg. 540, 8000 Aarhus C, Denmark
E-mail: mn@brics.dk

Branislav Rován
Comenius University, Department of Computer Science
84248 Bratislava, Slovakia
E-mail: rovan@fmph.uniba.sk

Cataloging-in-Publication Data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Mathematical foundations of computer science 2000 : 25th international symposium ; proceedings / MFCS 2000, Bratislava, Slovakia, August 28 - September 1, 2000. Mogens Nielsen ; Branislav Rován (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 2000
(Lecture notes in computer science ; Vol. 1893)
ISBN 3-540-67901-4

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

ISSN 0302-9743

ISBN 3-540-67901-4 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 PTP-Berlin, Stefan Sossna
Printed on acid-free paper SPIN: 10722549 06/3142 5 4 3 2 1 0

Foreword

This volume contains papers selected for presentation at the Silver Jubilee 25th Symposium on Mathematical Foundations of Computer Science — MFCS 2000, held in Bratislava, Slovakia, August 28 – September 1, 2000.

MFCS 2000 was organized under the auspices of the Minister of Education of the Slovak Republic, Milan Ftáčnik, by the Slovak Society for Computer Science, and the Comenius University in Bratislava, in cooperation with other institutions in Slovakia. It was supported by the European Association for Theoretical Computer Science, the European Research Consortium for Informatics and Mathematics, and the Slovak Research Consortium for Informatics and Mathematics.

The series of MFCS symposia, organized alternately in the Czech Republic, Poland, and Slovakia since 1972, has a well-established tradition. The MFCS symposia encourage high-quality research in all branches of theoretical computer science. Their broad scope provides an opportunity of bringing together specialists who do not usually meet at specialized conferences. The previous meetings took place in Jablonna, 1972; Štrbské Pleso, 1973; Jadwisin, 1974; Mariánské Lázně, 1975; Gdańsk, 1976; Tatranská Lomnica, 1977; Zakopane, 1978; Olomouc, 1979; Rydzina, 1980; Štrbské Pleso, 1981; Prague, 1984; Bratislava, 1986; Carlsbad, 1988; Porąbka-Kozubnik, 1989; Banská Bystrica, 1990; Kazimierz Dolny, 1991; Prague, 1992; Gdańsk, 1993; Košice, 1994; Prague, 1995; Kraków, 1996; Bratislava, 1997; Brno, 1998; and Szklarska Poreba, 1999.

The MFCS 2000 conference was accompanied by three satellite workshops taking place at the weekends preceding and following MFCS 2000. These were *Algorithmic Foundations of Communication Networks*, coordinated by M. Mavronicolas; *New Developments in Formal Languages versus Complexity*, coordinated by K.W. Wagner; and *Prague Stringology Club Workshop 2000*, coordinated by B. Melichar.

The MFCS 2000 proceedings consist of 8 invited papers and 57 contributed papers. The latter were selected by the Program Committee from a total of 147 submitted papers. The following program committee members took part in the evaluation and selection of submitted papers (those denoted by * took part at the selection meeting in Bratislava on May 20–21, 2000): M. Broy (Munich), J. Díaz (Barcelona), R. Freivalds (Riga), Z. Fülöp* (Szeged), G. Gottlob (Vienna), B. Jonsson (Uppsala), J. Karhumäki* (Turku), L. Kari (London, Ontario), D. Kozen (Ithaca), M. Křetínský* (Brno), C. Marché (Orsay), A. Marchetti-Spaccamela (Rome), M. Mavronicolas* (Nicosia), B. Monien* (Paderborn), M. Nielsen* (Aarhus, *co-chair*), L. Pacholski* (Wrocław), J.-E. Pin (Paris), B. Rován* (Bratislava, *chair*), J. Rutten* (Amsterdam), P. Ružička* (Bratislava), V. Sassone* (Catania), J. Sgall (Prague), A. Simpson* (Edinburgh), K. Wagner* (Würzburg), I. Walukiewicz* (Warsaw).

We would like to thank all program committee members for their meritorious work in evaluating the submitted papers as well as the following referees, who assisted the program committee members:

F. Ablayev, L. Aceto, M. Alberts, G. Alford, E. Allender, H. Alt, Th. Altenkirch, C. Alvarez, A. Ambainis, T. Amnell, F. d'Amore, O. Arieli, A. Arnold, D. Aspinall, A. Atserias, J.-M. Autebert, F. Baader, M. Baaz, M. Baldamus, P. Baldan, A. Baltag, G. Bauer, M. von der Beeck, P. Berenbrink, M. Bernardo, J.-C. Birget, F.S. de Boer, R. Bol, M. Bonsangue, S. Bozapalidis, J. Bradfield, P. Braun, M. Breitling, L. Brim, R. Bruni, M. Bruynooghe, H. Buhrman, J.-Y. Cai, C. Calcagno, C.S. Calude, J. Cassaigne, I. Černá, M. Chrobak, C. Cirstea, A. Clementi, A. Condon, A. Corradini, T. Crolard, R. Crole, J. Csima, E. Csuhaj-Varjú, A. Czumaj, M. Dakilic, M. Daley, G. Dányi, R. De Nicola, M. Dietzfelbinger, F. Drewes, J.-P. Duval, H. Ehler, S. Eidenbenz, R. Elsaesser, Z. Ésik, T. Fahle, P. Fatourou, R. Feldmann, Ch. Fermueller, J.-C. Filliâtre, M. Fiore, R. Focardi, L. Fortnow, D. Fridlender, D. Frigioni, C. Fritz, J. Gabarró, F. Gadducci, V. Geffert, J. Gehring, R. Glück, Ch. Glasser, I. Gnaedig, S. Goetz, B. Gramlich, S. Grothklags, V. Halava, A. Hall, T. Harju, J. den Hartog, M. Hasegawa, K. Havelund, J.G. Henriksen, H. Herbelin, M. Hermann, U. Hertrampf, Th. Hildebrandt, M. Hirvensalo, Th. Hofmeister, J. Honkala, H.J. Hoogeboom, J. Hromkovič, F. Hurtado, H. Hüttel, K. Inoue, P. Iyer, P. Jančar, T. Jiang, J.E. Jonker, M. Jurdzinski, T. Jurdzinski, B. König, S. Kahrs, J. Kaneps, J. Kari, Z. Kása, R. Khardon, Z. Khasidashvili, E. Kieroński, A. Kisielewicz, R. Kitto, M. Kiwi, Klein, G. Kliwer, J. Koebler, B. Konikowska, S. Kosub, J. Kristoffersen, I. Krüger, R. Kráľovič, J. Krajíček, M. Krause, A. Kučera, O. Kupferman, P. Kůrka, D. Kuske, M. Kutrib, R. Laemmel, C. Laneve, M. Latteux, Th. Lecroq, M. Lenisa, A. Lepisto, F. Levi, L. Libkin, L. Lisovic, I. Litovsky, H. Loetz-beyer, J. Longley, U. Lorenz, A. Lozano, R. Lueling, G. Luetzgen, M.O. Möller, J. Mañuch, A. Maes, J. Marcinkowski, E. Mayordomo, E.W. Mayr, R. Mayr, J. Mazoyer, M. McGettrick, W. Merkle, O. Miš, M. Miculan, G. Mirkowska, M. Mlotkowski, R. Mubarakzjanov, M. Mundhenk, A. Muscholl, W. Naraschewski, P. Narendran, G. Navarro, U. Nestmann, M. Nilsson, C. Nippl, D. Niwinski, A. Nonnengart, A. Nylén, D. von Oheimb, H.J. Ohlbach, V. van Oostrom, M. Parigot, R. Pendavingh, T. Petkovic, I. Petre, P. Pettersson, A. Philippou, Philipps, A. Piperno, M. Pistore, T. Plachetka, M. Pocchiola, J. Power, R. Preis, L. Prensa, A. Pretschner, P. Pudlák, R. Pugliese, A. Pultr, Ch. Röckl, P. Réty, J. Ramon, K.W. Regan, M. Riedel, P. Rychlikowski, Z. Sadowski, A. Salomaa, G. Salzer, R. Sandner, D. Sangiorgi, J.E. Santo, L. Santocanale, P. Savicky, B. Schätz, Ch. Scheideler, T. Schickinger, B. Schieder, A.B. Schmidt, O. Schmidt, H. Schmitz, U. Schoening, M. Schoenmakers, U.-P. Schroeder, R. Schuler, J. Schulze, R.A.G. Seely, M. Sellmann, A.L. Selman, N. Sensen, M. Serna, P. Sewell, H. Sezinando, A. Shokrollahi, J. Simpson, L. Škarvada, A. Slobodová, O. Sokolsky, K. Spies, Z. Splawski, J. Srba, J. Štříbrná, L. Staiger, Th. Stauner, R. Steinbrueggen, C. Stirling, L. Stougie, H. Straubing, A. Streit, D. Taimina, V. Terrier, P. Thanisch, P.S. Thiagarajan, Th. Thierauf, W. Tomanik, J. Toran, T. Truderung, E. Ukkonen, P. Valtr, Gy. Vaszil, H. Veith,

B. Victor, L. Viganò, J. Vogel, S. Vogel, H. Vollmer, S. Vorobyov, R. Wanka, O. Watanabe, M. Wenzel, P. Widmayer, T. Wierzbicki, T. Wilke, G. Winskel, G. Woeginger, R. de Wolf, U. Wolter, J. Worrell, D. Wotschke, W. Zielonka.

Being the editors of these proceedings we are much indebted to all contributors to the scientific program of the symposium, especially to the authors of papers. Special thanks go to those authors who prepared the manuscripts according to the instructions and made life easier for us. We would also like to thank those who responded promptly to our requests for minor modifications and corrections in their manuscript. Our special thanks belong to Miroslav Chladný who designed (and manned) the database and electronic support for the Program Committee and who did most of the hard technical work in preparing this volume. We are also thankful to the members of the Organizing Committee who made sure that the conference ran smoothly in a pleasant environment. Last but not least we want to thank Springer-Verlag for excellent co-operation in publication of this volume.

June 2000

Mogens Nielsen, Branislav Rován



SILVER JUBILEE 25th MFCS Conference

Organized under the auspices of

the Minister of Education of the Slovak Republic Milan Ftáčnik

by

Slovak Society for Computer Science

Faculty of Mathematics and Physics, Comenius University, Bratislava

Supported by

European Association for Theoretical Computer Science

European Research Consortium for Informatics and Mathematics

Slovak Research Consortium for Informatics and Mathematics



Telenor Slovakia provided Internet connection to the conference site and hosted the MFCS 2000 web page.

Program Committee

M. Broy (Munich), J. Díaz (Barcelona), R. Freivalds (Riga), Z. Fülöp (Szeged), G. Gottlob (Vienna), B. Jonsson (Uppsala), J. Karhumäki (Turku), L. Kari (London, Ontario), D. Kozen (Ithaca), M. Křetínský (Brno), C. Marché (Orsay), A. Marchetti-Spaccamela (Rome), M. Mavronicolas (Nicosia), B. Monien (Paderborn), M. Nielsen (Aarhus, *co-chair*), L. Pacholski (Wrocław), J.-E. Pin (Paris), B. Rovan (Bratislava, *chair*), J. Rutten (Amsterdam), P. Ružička (Bratislava), V. Sassone (Catania), J. Sgall (Prague), A. Simpson (Edinburgh), K. Wagner (Würzburg), I. Walukiewicz (Warsaw)

Organizing Committee

Martin Bečka, Miroslav Chladný, Rastislav Graus, Vanda Hambáľková, Zuzana Kubincová, Martin Nehéz, Marek Nagy, Dana Pardubská (*vice-chair*), Edita Ričányová, Branislav Rovan (*chair*)

Table of Contents

Invited Talks

Region Analysis and a π -Calculus with Groups	1
<i>Silvano Dal Zilio and Andrew D. Gordon</i>	
Abstract Data Types in Computer Algebra.....	21
<i>James H. Davenport</i>	
What Do We Learn from Experimental Algorithmics?	36
<i>Camil Demetrescu and Giuseppe F. Italiano</i>	
And/Or Hierarchies and Round Abstraction	52
<i>Radu Grosu</i>	
Computational Politics: Electoral Systems.....	64
<i>Edith Hemaspaandra and Lane A. Hemaspaandra</i>	
0-1 Laws for Fragments of Existential Second-Order Logic: A Survey	84
<i>Phokion G. Kolaitis and Moshe Y. Vardi</i>	
On Algorithms and Interaction	99
<i>Jan van Leeuwen and Jiří Wiedermann</i>	
On the Use of Duality and Geometry in Layouts for ATM Networks	114
<i>Shmuel Zaks</i>	

Contributed Papers

On the Lower Bounds for One-Way Quantum Automata	132
<i>Farid Ablayev and Aida Gainutdinova</i>	
Axiomatizing Fully Complete Models for ML Polymorphic Types	141
<i>Samson Abramsky and Marina Lenisa</i>	
Measure Theoretic Completeness Notions for the Exponential Time Classes	152
<i>Klaus Ambos-Spies</i>	
Edge-Bisection of Chordal Rings	162
<i>Lali Barrière and Josep Fàbrega</i>	
Equation Satisfiability and Program Satisfiability for Finite Monoids	172
<i>David Mix Barrington, Pierre McKenzie, Cris Moore, Pascal Tesson, and Denis Thérien</i>	

XML Grammars	182
<i>Jean Berstel and Luc Boasson</i>	
Simplifying Flow Networks	192
<i>Therese C. Biedl, Broňa Brejová, and Tomáš Vinař</i>	
Balanced k -Colorings	202
<i>Therese C. Biedl, Eowyn Čenek, Timothy M. Chan, Erik D. Demaine, Martin L. Demaine, Rudolf Fleischer, and Ming-Wei Wang</i>	
A Compositional Model for Confluent Dynamic Data-Flow Networks	212
<i>Frank S. de Boer and Marcello M. Bonsangue</i>	
Restricted Nondeterministic Read-Once Branching Programs and an Exponential Lower Bound for Integer Multiplication (Extended Abstract) .	222
<i>Beate Bollig</i>	
Expressiveness of Updatable Timed Automata	232
<i>P. Bouyer, C. Dufourd, E. Fleury, and A. Petit</i>	
Iterative Arrays with Small Time Bounds	243
<i>Thomas Buchholz, Andreas Klein, and Martin Kutrib</i>	
Embedding Fibonacci Cubes into Hypercubes with $\Omega(2^{cn})$ Faulty Nodes . .	253
<i>Rostislav Caha and Petr Gregor</i>	
Periodic-Like Words	264
<i>Arturo Carpi and Aldo de Luca</i>	
The Monadic Theory of Morphic Infinite Words and Generalizations	275
<i>Olivier Carton and Wolfgang Thomas</i>	
Optical Routing of Uniform Instances in Tori	285
<i>Francesc Comellas, Margarida Mitjana, Lata Narayanan, and Jaroslav Opatrny</i>	
Factorizing Codes and Schützenberger Conjectures	295
<i>Clelia De Felice</i>	
Compositional Characterizations of λ -Terms Using Intersection Types (Extended Abstract)	304
<i>M. Dezani-Ciancaglini, F. Honsell, and Y. Motohama</i>	
Time and Message Optimal Leader Election in Asynchronous Oriented Complete Networks	314
<i>Stefan Dobrev</i>	
Subtractive Reductions and Complete Problems for Counting Complexity Classes	323
<i>Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis</i>	

On the Autoreducibility of Random Sequences	333
<i>Todd Ebert and Heribert Vollmer</i>	
Iteration Theories of Boolean Functions	343
<i>Zoltán Ésik</i>	
An Algorithm Constructing the Semilinear Post* for 2-Dim Reset/Transfer VASS (Extended Abstract)	353
<i>A. Finkel and G. Sutre</i>	
NP-Completeness Results and Efficient Approximations for Radiocoloring in Planar Graphs	363
<i>D.A. Fotakis, S.E. Nikolettseas, V.G. Papadopoulou, and P.G. Spirakis</i>	
Explicit Fusions	373
<i>Philippa Gardner and Lucian Wischik</i>	
State Space Reduction Using Partial τ -Confluence	383
<i>Jan Friso Groote and Jaco van de Pol</i>	
Reducing the Number of Solutions of NP Functions	394
<i>Lane A. Hemaspaandra, Mitsunori Ogihara, and Gerd Wechsung</i>	
Regular Collections of Message Sequence Charts (Extended Abstract)	405
<i>Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, and P.S. Thiagarajan</i>	
Alternating and Empty Alternating Auxiliary Stack Automata	415
<i>Markus Holzer and Pierre McKenzie</i>	
Counter Machines: Decidable Properties and Applications to Verification Problems	426
<i>Oscar H. Ibarra, Jianwen Su, Zhe Dang, Tefvik Bultan, and Richard Kemmerer</i>	
A Family of NFA's Which Need $2^n - \alpha$ Deterministic States	436
<i>Kazuo Iwama, Akihiro Matsuura, and Mike Paterson</i>	
Preemptive Scheduling on Dedicated Processors: Applications of Fractional Graph Coloring	446
<i>Klaus Jansen and Lóránt Porkolab</i>	
Matching Modulo Associativity and Idempotency Is NP-Complete	456
<i>Ondřej Klíma and Jiří Srba</i>	
On NP-Partitions over Posets with an Application to Reducing the Set of Solutions of NP Problems	467
<i>Sven Kosub</i>	

Algebraic and Uniqueness Properties of Parity Ordered Binary Decision Diagrams and Their Generalization (Extended Abstract)	477
<i>Daniel Král'</i>	
Formal Series over Algebras	488
<i>Werner Kuich</i>	
μ -Calculus Synthesis	497
<i>Orna Kupferman and Moshe Y. Vardi</i>	
The Infinite Versions of $\text{LOGSPACE} \neq \text{P}$ Are Consistent with the Axioms of Set Theory	508
<i>Grégory Lafitte and Jacques Mazoyer</i>	
Timed Automata with Monotonic Activities	518
<i>Ruggero Lanotte and Andrea Maggiolo-Schettini</i>	
On a Generalization of Bi-Complement Reducible Graphs	528
<i>Vadim V. Lozin</i>	
Automatic Graphs and Graph $D0L$ -Systems	539
<i>Olivier Ly</i>	
Bilinear Functions and Trees over the $(\max, +)$ Semiring	549
<i>Sabrina Mantaci, Vincent D. Blondel, and Jean Mairesse</i>	
Derivability in Locally Quantified Modal Logics via Translation in Set Theory	559
<i>Angelo Montanari, Alberto Policriti, and Matteo Slanina</i>	
π -Calculus, Structured Coalgebras, and Minimal HD-Automata	569
<i>Ugo Montanari and Marco Pistore</i>	
Informative Labeling Schemes for Graphs (Extended Abstract)	579
<i>David Peleg</i>	
Separation Results for Rebound Automata	589
<i>Holger Petersen</i>	
Unary Pushdown Automata and Auxiliary Space Lower Bounds	599
<i>Giovanni Pighizzini</i>	
Binary Decision Diagrams by Shared Rewriting	609
<i>Jaco van de Pol and Hans Zantema</i>	
Verifying Single and Multi-mutator Garbage Collectors with Owicki-Gries in Isabelle/HOL	619
<i>Leonor Prensa Nieto and Javier Esparza</i>	

Why so Many Temporal Logics Climb up the Trees?	629
<i>Alexander Rabinovich and Shahar Maoz</i>	
Optimal Satisfiability for Propositional Calculi and Constraint Satisfaction Problems	640
<i>Steffen Reith and Heribert Vollmer</i>	
A Hierarchy Result for Read-Once Branching Programs with Restricted Parity Nondeterminism (Extended Abstract)	650
<i>Petr Savický and Detlef Sieling</i>	
On Diving in Trees	660
<i>Thomas Schwentick</i>	
Abstract Syntax and Variable Binding for Linear Binders	670
<i>Miki Tanaka</i>	
Regularity of Congruential Graphs	680
<i>Tanguy Urvoy</i>	
Sublinear Ambiguity	690
<i>Klaus Wich</i>	
An Automata-Based Recognition Algorithm for Semi-extended Regular Expressions	699
<i>Hiroaki Yamamoto</i>	
Author Index	709

Region Analysis and a π -Calculus with Groups

Silvano Dal Zilio and Andrew D. Gordon

Microsoft Research

Abstract. We show that the typed region calculus of Tofte and Talpin can be encoded in a typed π -calculus equipped with name groups and a novel effect analysis. In the region calculus, each boxed value has a statically determined region in which it is stored. Regions are allocated and de-allocated according to a stack discipline, thus improving memory management. The idea of name groups arose in the typed ambient calculus of Cardelli, Ghelli, and Gordon. There, and in our π -calculus, each name has a statically determined group to which it belongs. Groups allow for type-checking of certain mobility properties, as well as effect analyses. Our encoding makes precise the intuitive correspondence between regions and groups. We propose a new formulation of the type preservation property of the region calculus, which avoids Tofte and Talpin's rather elaborate co-inductive formulation. We prove the encoding preserves the static and dynamic semantics of the region calculus. Our proof of the correctness of region de-allocation shows it to be a specific instance of a general garbage collection principle for the π -calculus with effects.

1 Motivation

This paper reports a new proof of correctness of region-based memory management [26], based on a new garbage collection principle for the π -calculus.

Tofte and Talpin's region calculus is a compiler intermediate language that, remarkably, supports an implementation of Standard ML that has no garbage collector, the ML Kit compiler [4]. The basic idea of the region calculus is to partition heap memory into a stack of regions. Each boxed value (that is, a heap-allocated value such as a closure or a cons cell) is annotated with the particular region into which it is stored. The construct *letregion* ρ in b manages the allocation and de-allocation of regions. It means: "Allocate a fresh, empty region, denoted by the region variable ρ ; evaluate the expression b ; de-allocate ρ ." A type and effect system for the region calculus guarantees the safety of de-allocating the defunct region as the last step of *letregion*. The allocation and de-allocation of regions obeys a stack discipline determined by the nesting of the *letregion* constructs. A region inference algorithm compiles ML to the region calculus by computing suitable region annotations for boxed values, and inserting *letregion* constructs as necessary. In practice, space leaks, where a particular region grows without bound, are a problem. Still, they can practically always be detected by profiling and eliminated by simple modifications. The ML Kit efficiently executes an impressive range of benchmarks without a garbage

collector and without space leaks. Region-based memory management facilitates interoperability with languages like C that have no garbage collector and helps enable realtime applications of functional programming.

Tofte and Talpin’s semantics of the region calculus is a structural operational semantics. A map from region names to their contents represents the heap. A fresh region name is invented on each evaluation of *letregion*. This semantics supports a co-inductive proof of type safety, including the safety of de-allocating the defunct region at the end of each *letregion*. The proof is complex and surprisingly subtle, in part because active regions may contain dangling pointers that refer to de-allocated regions.

The region calculus is a strikingly simple example of a language with type generativity. A language has type generativity when type equivalence is by name (that is, when types with different names but the same structure are not equivalent), and when type names can be generated at run-time. A prominent example is the core of Standard ML [17], whose *datatype* construct effectively generates a fresh algebraic type each time it is evaluated. (The ML module system also admits type generativity, but at link-time rather than run-time.) The region calculus has type generativity because the type of a boxed value includes the name of the region where it lives, and region names are dynamically generated by *letregion*. The semantics of Standard ML accounts operationally for type generativity by inventing a fresh type name on each elaboration of *datatype*. Various researchers have sought more abstract accounts of type generativity [13,21].

This paper describes a new semantics for a form of the region calculus, obtained by translation to a typed π -calculus equipped with a novel effect system. The π -calculus [15] is a rather parsimonious formalism for describing the essential semantics of concurrent systems. It serves as a foundation for describing a variety of imperative, functional, and object-oriented programming features [22,25,28], for the design of concurrent programming languages [9,20], and for the study of security protocols [1], as well as other applications. The only data in the π -calculus are atomic names. Names can model a wide variety of identifiers: communication channels, machine addresses, pointers, object references, cryptographic keys, and so on. A new-name construct $(\nu x)P$ generates names dynamically in the standard π -calculus. It means: “Invent a fresh name, denoted by x ; run process P .” One might hope to model region names with π -calculus names but unfortunately typings would not be preserved: a region name may occur in a region-calculus type, but in standard typed π -calculi [19], names may not occur in types.

We solve the problem of modelling region names by defining a typed π -calculus equipped with name groups and a new-group construct [5]. The idea is that each π -calculus name belongs to a group, G . The type of a name now includes its group. A new-group construct $(\nu G)P$ generates groups dynamically. It means: “Invent a fresh group, denoted by G ; run process P .” The basic ideas of the new semantics are that region names are groups, that pointers into a region ρ are names of group ρ , and that given a continuation channel k the continuation-passing semantics of *letregion* ρ in b is simply the process $(\nu \rho)[b]k$

where $\llbracket b \rrbracket k$ is the semantics of expression b . The semantics of other expressions is much as in earlier π -calculus semantics of λ -calculi [22]. Parallelism allows us to explain a whole functional computation as an assembly of individual processes that represent components such as closures, continuations, and function invocations.

This new semantics for regions makes two main contributions.

- First, we give a new proof of the correctness of memory management in the region calculus. We begin by extending a standard encoding with the equation $\llbracket \text{letregion } \rho \text{ in } b \rrbracket k = (\nu \rho) \llbracket b \rrbracket k$. Then the rather subtle correctness property of de-allocation of defunct regions turns out to be a simple instance of a new abstract principle expressed in the π -calculus. Hence, an advantage of our π -calculus proof is that it is conceptually simpler than a direct proof.
- Second, the semantics provides a more abstract, equational account of type generativity in the region calculus than the standard operational semantics.

The specific technical results of the paper are:

- A simple proof of type soundness of the region calculus (Theorem 1).
- A new semantics of the region calculus in terms of the π -calculus with groups. The translation preserves types and effects (Theorem 2) and operational behaviour (Theorem 3).
- A new garbage collection principle for the π -calculus (Theorem 4) whose corollary (Theorem 5) justifies de-allocation of defunct regions in the region calculus.

We organise the rest of the paper as follows. Section 2 introduces the region calculus. Section 3 describes the π -calculus with groups and effects. Section 4 gives our new π -calculus semantics for regions. Section 5 concludes. Omitted proofs may be found in a long version of this paper [8].

2 A λ -Calculus with Regions

To focus on the encoding of *letregion* with the new-group construct, we work with a simplified version of the region calculus of Tofte and Talpin [26]. Our calculus omits the recursive functions, type polymorphism, and region polymorphism present in Tofte and Talpin’s calculus. The long version of this paper includes an extension of our results to a region calculus with recursive functions, finite lists, and region polymorphism. To encode these features, we need to extend our π -calculus with recursive types and group polymorphism. Tofte and Talpin explain that type polymorphism is not essential for their results. Still, we conjecture that our framework could easily accommodate type polymorphism.

2.1 Syntax

Our region calculus is a typed call-by-value λ -calculus equipped with a *letregion* construct and an annotation on each function to indicate its storage region. We

assume an infinite set of *names*, ranged over by p, q, x, y, z . For the sake of simplicity, names represent both program variables and memory pointers, and a subset of the names $L = \{\ell_1, \dots, \ell_n\}$ represents literals. The following table defines the syntax of λ -calculus expressions, a or b , as well as an auxiliary notion of boxed value, u or v .

Expressions and Values:

x, y, p, q, f, g	name: variable, pointer, literal
ρ	region variable
$a, b ::=$	expression
x	name
$v \text{ at } \rho$	allocation of v at ρ
$x(y)$	application
$\text{let } x = a \text{ in } b$	sequencing
$\text{letregion } \rho \text{ in } b$	region allocation, de-allocation
$u, v ::=$	boxed value
$\lambda(x:A)b$	function

We shall explain the type A later. In both $\text{let } x = a \text{ in } b$ and $\lambda(x:A)b$, the name x is bound with scope b . Let $fn(a)$ be the set of names that occur free in the expression a . We identify expressions and values up to consistent renaming of bound names. We write $P\{x \leftarrow y\}$ for the outcome of renaming all free occurrences of x in P to the name y . Our syntax is in a reduced form, where an application $x(y)$ is of a name to a name. We can regard a conventional application $b(a)$ as an abbreviation for $\text{let } f = b \text{ in } \text{let } x = a \text{ in } f(x)$, where $f \neq x$ and f is not free in a .

We explain the intended meaning of the syntax by example. The following expression,

$$\begin{aligned}
 ex_1 &\triangleq \text{letregion } \rho' \text{ in} \\
 &\quad \text{let } f = \lambda(x:Lit)x \text{ at } \rho' \text{ in} \\
 &\quad \text{let } g = \lambda(y:Lit)f(y) \text{ at } \rho \text{ in } g(5)
 \end{aligned}$$

means: “Allocate a fresh, empty region, and bind it to ρ' ; allocate $\lambda(x:Lit)x$ in region ρ' , and bind the pointer to f ; allocate $\lambda(y:Lit)f(y)$ in region ρ (an already existing region), and bind the pointer to g ; call the function at g with literal argument 5; finally, de-allocate ρ' .” The function call amounts to calling $\lambda(y:Lit)f(y)$ with argument 5. So we call $\lambda(x:Lit)x$ with argument 5, which immediately returns 5. Hence, the final outcome is the answer 5, and a heap containing a region ρ with g pointing to $\lambda(y:Lit)f(y)$. The intermediate region ρ' has gone. Any subsequent invocations of the function $\lambda(y:Lit)f(y)$ would go wrong, since the target of f has been de-allocated. The type and effect system of Section 2.3 guarantees there are no subsequent allocations or invocations on region ρ' , such as invoking $\lambda(y:Lit)f(y)$.

2.2 Dynamic Semantics

Like Tofte and Talpin, we formalize the intuitive semantics via a conventional structural operational semantics. A heap, h , is a map from region names to regions, and a region, r , is a map from pointers (names) to boxed values (function closures). In Tofte and Talpin's semantics, defunct regions are erased from the heap when they are de-allocated. In our semantics, the heap consists of both live regions and defunct regions. Our semantics maintains a set S containing the region names for the live regions. This is the main difference between the two semantics. Side-conditions on the evaluation rules guarantee that only the live regions in S are accessed during evaluation. Retaining the defunct regions simplifies the proof of subject reduction. Semmelroth and Sabry [23] adopt a similar technique for the same reason in their semantics of monadic encapsulation.

Regions, Heaps, and Stacks:

$r ::= (p_i \mapsto v_i)_{i \in 1..n}$	region, p_i distinct
$h ::= (\rho_i \mapsto r_i)_{i \in 1..n}$	heap, ρ_i distinct
$S ::= \{\rho_1, \dots, \rho_n\}$	stack of live regions

A region r is a finite map of the form $p_1 \mapsto v_1, \dots, p_n \mapsto v_n$, where the p_i are distinct, which we usually denote by $(p_i \mapsto v_i)_{i \in 1..n}$. An application, $r(p)$, of the map r to p denotes v_i , if p is p_i for some $i \in 1..n$. Otherwise, the application is undefined. The domain, $dom(r)$, of the map r is the set $\{p_1, \dots, p_n\}$. We write $\emptyset$ for the empty map. If $r = (p_i \mapsto v_i)_{i \in 1..n}$, we define the notation $r - p$ to be $p_i \mapsto v_i_{i \in (1..n) - \{j\}}$ if $p = p_j$ for some $j \in 1..n$, and otherwise to be simply r . Then we define the notation $r + (p \mapsto v)$ to mean $(r - p), p \mapsto v$.

We use finite maps to represent regions, but also heaps, and various other structures. The notational conventions defined above for regions apply also to other finite maps, such as heaps. Additionally, we define $dom_2(h)$ to be the set of all pointers defined in h , that is, $\bigcup_{\rho \in dom(h)} dom(h(\rho))$.

The evaluation relation, $S \cdot (a, h) \Downarrow (p, h')$, may be read: in an initial heap h , with live regions S , the expression a evaluates to the name p (a pointer or literal), leaving an updated heap h' , with the same live regions S .

Judgments:

$S \cdot (a, h) \Downarrow (p, h')$	evaluation

Evaluation Rules:

(Eval Var)	(Eval Alloc)
	$\rho \in S \quad p \notin dom_2(h)$
$S \cdot (p, h) \Downarrow (p, h)$	$S \cdot (v \text{ at } \rho, h) \Downarrow (p, h + (\rho \mapsto (h(\rho) + (p \mapsto v))))$
(Eval Appl)	
$\rho \in S \quad h(\rho)(p) = \lambda(x:A)b$	$S \cdot (b\{x \leftarrow q\}, h) \Downarrow (p', h')$
	$S \cdot (p(q), h) \Downarrow (p', h')$

$$\begin{array}{c}
\text{(Eval Let)} \\
\frac{S \cdot (a, h) \Downarrow (p', h') \quad S \cdot (b\{x \leftarrow p'\}, h') \Downarrow (p'', h'')}{S \cdot (\text{let } x = a \text{ in } b, h) \Downarrow (p'', h'')} \\
\\
\text{(Eval Letregion)} \\
\frac{\rho \notin \text{dom}(h) \quad S \cup \{\rho\} \cdot (a, h + \rho \mapsto \emptyset) \Downarrow (p', h')}{S \cdot (\text{letregion } \rho \text{ in } a, h) \Downarrow (p', h')}
\end{array}$$

Recall the example expression ex_1 from the previous section. Consider an initial heap $h = \rho \mapsto \emptyset$ and a region stack $S = \{\rho\}$, together representing a heap with a single region ρ that is live but empty. We can derive $S \cdot (ex_1, h) \Downarrow (5, h')$ where $h' = \rho \mapsto (g \mapsto \lambda(y:Lit)f(y)), \rho' \mapsto (f \mapsto \lambda(x:Lit)x)$. Since $\rho \in S$ but $\rho' \notin S$, ρ is live but ρ' is defunct.

2.3 Static Semantics

The static semantics of the region calculus is a simple type and effect system [10, 24, 27]. The central typing judgment of the static semantics is:

$$E \vdash a : \{\rho_1, \dots, \rho_n\} A$$

which means that in a typing environment E , the expression a may yield a result of type A , while allocating and invoking boxed values stored in regions $\rho_1, \dots, \rho_n$. The set of regions $\{\rho_1, \dots, \rho_n\}$ is the *effect* of the expression, a bound on the interactions between the expression and the store. For simplicity, we have dropped the distinction between allocations, $put(\rho)$, and invocations, $get(\rho)$, in Tofte and Talpin's effects. This is an inessential simplification; the distinction could easily be added to our work.

An expression type, A , is either *Lit*, a type of literal constants, or $(A \xrightarrow{e} B) \text{ at } \rho$, the type of a function stored in region ρ . The effect e is the latent effect: the effect unleashed by calling the function. An environment E has entries for the regions and names currently in scope.

Effects, Types, and Environments:

$e ::= \{\rho_1, \dots, \rho_n\}$	effect
$A, B ::=$	type of expressions
<i>Lit</i>	type of literals
$(A \xrightarrow{e} B) \text{ at } \rho$	type of functions stored in ρ
$E ::=$	environment
$\emptyset$	empty environment
E, ρ	entry for a region ρ
$E, x:A$	entry for a name x

Let $fr(A)$ be the set of region variables occurring in the type A . We define the domain, $\text{dom}(E)$, of an environment, E , by the equations $\text{dom}(\emptyset) = \emptyset$, $\text{dom}(E, \rho) = \text{dom}(E) \cup \{\rho\}$, and $\text{dom}(E, x:A) = \text{dom}(E) \cup \{x\}$.

The following tables present our type and effect system as a collection of typing judgments defined by a set of rules. Tofte and Talpin present their type and effect system in terms of constructing a region-annotated expression from an unannotated expression. Instead, our main judgment simply expresses the type and effect of a single region-annotated expression. Otherwise, our system is essentially the same as Tofte and Talpin's.

Type and Effect Judgments:

$E \vdash \diamond$	good environment
$E \vdash A$	good type
$E \vdash a :^e A$	good expression, with type A and effect e

Type and Effect Rules:

(Env $\emptyset$)	(Env x) (recall L is the set of literals)	(Env ρ)
$\frac{}{\emptyset \vdash \diamond}$	$\frac{E \vdash A \quad x \notin \text{dom}(E) \cup L}{E, x:A \vdash \diamond}$	$\frac{E \vdash \diamond \quad \rho \notin \text{dom}(E)}{E, \rho \vdash \diamond}$
(Type <i>Lit</i>)	(Type $\rightarrow$)	(Exp x)
$\frac{E \vdash \diamond}{E \vdash \textit{Lit}}$	$\frac{E \vdash A \quad \rho \cup \{e\} \subseteq \text{dom}(E) \quad E \vdash B}{E \vdash (A \xrightarrow{e} B) \text{ at } \rho}$	$\frac{E, x:A, E' \vdash \diamond}{E, x:A, E' \vdash x :^\emptyset A}$
(Exp ℓ)	(Exp Appl)	
$\frac{E \vdash \diamond \quad \ell \in L}{E \vdash \ell :^\emptyset \textit{Lit}}$	$\frac{E \vdash x :^\emptyset (B \xrightarrow{e} A) \text{ at } \rho \quad E \vdash y :^\emptyset B}{E \vdash x(y) :^{\{\rho\} \cup e} A}$	
(Exp Let)	(Exp Letregion)	
$\frac{E \vdash a :^e A \quad E, x:A \vdash b :^{e'} B}{E \vdash \textit{let } x = a \text{ in } b :^{e \cup e'} B}$	$\frac{E, \rho \vdash a :^e A \quad \rho \notin \text{fr}(A)}{E \vdash \textit{letregion } \rho \text{ in } a :^{e - \{\rho\}} A}$	
(Exp Fun)		
$\frac{E, x:A \vdash b :^e B \quad e \subseteq e' \quad \{\rho\} \cup e' \subseteq \text{dom}(E)}{E \vdash \lambda(x:A)b \text{ at } \rho :^{\{\rho\}} (A \xrightarrow{e'} B) \text{ at } \rho}$		

The rules for good environments are standard; they assure that all the names and region variables in the environment are distinct, and that the type of each name is good. All the regions in a good type must be declared. The type of a good expression is checked much as in the simply typed λ -calculus. The effect of a good expression is the union of all the regions in which it allocates or from which it invokes a closure. In the rule (Exp Letregion), the condition $\rho \notin \text{fr}(A)$ ensures that no function with a latent effect on the region ρ may be returned. Calling such a function would be unsafe since ρ is de-allocated once the *letregion* terminates. In the rule (Exp Fun), the effect e of the body of a function must be contained in the latent effect e' of the function. For the sake of simplicity we have no rule of effect subsumption, but it would be sound to add it: if $E \vdash a :^e A$

and $e' \subseteq \text{dom}(E)$ then $E \vdash a :^{e \cup e'} A$. In the presence of effect subsumption we could simplify (Exp Fun) by taking $e = e'$.

Recall the expression ex_1 from Section 2.1. We can derive the following:

$$\begin{aligned} \rho, \rho' &\vdash (\lambda(x:\text{Lit})x) \text{ at } \rho' :^{\{\rho'\}} (\text{Lit} \xrightarrow{\varnothing} \text{Lit}) \text{ at } \rho' \\ \rho, \rho', f &: (\text{Lit} \xrightarrow{\varnothing} \text{Lit}) \text{ at } \rho' \vdash (\lambda(x:\text{Lit})f(x)) \text{ at } \rho :^{\{\rho\}} (\text{Lit} \xrightarrow{\{\rho'\}} \text{Lit}) \text{ at } \rho \\ \rho, \rho', f &: (\text{Lit} \xrightarrow{\varnothing} \text{Lit}) \text{ at } \rho', g : (\text{Lit} \xrightarrow{\{\rho'\}} \text{Lit}) \text{ at } \rho \vdash g(5) :^{\{\rho, \rho'\}} \text{Lit} \end{aligned}$$

Hence, we can derive $\rho \vdash ex_1 :^{\{\rho\}} \text{Lit}$.

For an example of a type error, suppose we replace the application $g(5)$ in ex_1 simply with the identifier g . Then we cannot type-check the *letregion* ρ' construct, because ρ' is free in the type of its body. This is just as well, because otherwise we could invoke a function in a defunct region.

For an example of how a dangling pointer may be passed around harmlessly, but not invoked, consider the following. Let F abbreviate the type $(\text{Lit} \xrightarrow{\varnothing} \text{Lit}) \text{ at } \rho'$. Let ex_2 be the following expression:

$$\begin{aligned} ex_2 &\triangleq \text{letregion } \rho' \text{ in} \\ &\quad \text{let } f = \lambda(x:\text{Lit})x \text{ at } \rho' \text{ in} \\ &\quad \text{let } g = \lambda(f:F)5 \text{ at } \rho \text{ in} \\ &\quad \text{let } j = \lambda(z:\text{Lit})g(f) \text{ at } \rho \text{ in } j \end{aligned}$$

We have $\rho \vdash ex_2 :^{\{\rho\}} (\text{Lit} \xrightarrow{\{\rho\}} \text{Lit}) \text{ at } \rho$. If $S = \{\rho\}$ and $h = \rho \mapsto \varnothing$, then $S \cdot (b, h) \Downarrow (j, h')$ where the final heap h' is $\rho \mapsto (g \mapsto \lambda(f:F)5, j \mapsto \lambda(z:\text{Lit})g(f)), \rho' \mapsto (f \mapsto \lambda(x:\text{Lit})x)$. In the final heap, there is a pointer f from the live region ρ to the defunct region ρ' . Whenever j is invoked, this pointer will be passed to g , harmlessly, since g will not invoke it.

2.4 Relating the Static and Dynamic Semantics

To relate the static and dynamic semantics, we need to define when a configuration is well-typed. First, we need notions of region and heap typings. A region typing R tracks the types of boxed values in the region. A heap typing H tracks the region typings of all the regions in a heap. The environment $\text{env}(H)$ lists all the regions in H , followed by types for all the pointers in those regions.

Region and Heap Typings:

$R ::= (p_i:A_i) \ i \in 1..n$	region typing
$H ::= (\rho_i \mapsto R_i) \ i \in 1..n$	heap typing
$\text{ptr}(H) \triangleq R_1, \dots, R_n$	if $H = (\rho_i \mapsto R_i) \ i \in 1..n$
$\text{env}(H) \triangleq \text{dom}(H), \text{ptr}(H)$	

The next tables describe the judgments and rules defining well-typed regions, heaps, and configurations. The main judgment $H \models S \cdot (a, h) : A$ means that a configuration $S \cdot (a, h)$ is well-typed: the heap h conforms to H and the expression a returns a result of type A , and its effect is within the live regions S .

Region, Heap, and Configuration Judgments:

$E \vdash r \text{ at } \rho : R$	in E , region r , named ρ , has type R
$H \models \diamond$	the heap typing H is good
$H \models h$	in H , the heap h is good
$H \models S \cdot (a, h) : A$	in H , configuration $S \cdot (a, h)$ returns A

Region, Heap, and Configuration Rules:

(Region Good)

$$\frac{E \vdash v_i \text{ at } \rho : \{\rho\} A_i \quad \forall i \in 1..n}{E \vdash (p_i \mapsto v_i)^{i \in 1..n} \text{ at } \rho : (p_i : A_i)^{i \in 1..n}}$$

(Heap Typing Good)

$$\frac{env(H) \vdash \diamond}{H \models \diamond}$$

(Heap Good) (where $dom(H) = dom(h)$)

$$\frac{env(H) \vdash h(\rho) \text{ at } \rho : H(\rho) \quad \forall \rho \in dom(H)}{H \models h}$$

(Config Good) (where $S \subseteq dom(H)$)

$$\frac{env(H) \vdash a :^e A \quad e \cup fr(A) \subseteq S \quad H \models h}{H \models S \cdot (a, h) : A}$$

These predicates roughly correspond to the co-inductively defined consistency predicate of Tofte and Talpin. The retention of defunct regions in our semantics allows a simple inductive definition of these predicates, and a routine inductive proof of the subject reduction theorem stated below.

We now present a subject reduction result relating the static and dynamic semantics. Let $H \asymp H'$ if and only if the pointers defined by H and H' are disjoint, that is, $dom_2(H) \cap dom_2(H') = \emptyset$. Assuming that $H \asymp H'$, we write $H + H'$ for the heap consisting of all the regions in either H or H' ; if ρ is in both heaps, $(H + H')(\rho)$ is the concatenation of the two regions $H(\rho)$ and $H'(\rho)$.

Theorem 1. *If $H \models S \cdot (a, h) : A$ and $S \cdot (a, h) \Downarrow (p', h')$ there is H' such that $H \asymp H'$ and $H + H' \models S \cdot (p', h') : A$.*

Intuitively, the theorem asserts that evaluation of a well-typed configuration $S \cdot (a, h)$ leads to another well-typed configuration $S \cdot (p', h')$, where H' represents types for the new pointers and regions in h' .

The following proposition shows that well-typed configurations avoid the run-time errors of allocation or invocation of a closure in a defunct region.

Proposition 1.

- (1) *If $H \models S \cdot (v \text{ at } \rho, h) : A$ then $\rho \in S$.*
- (2) *If $H \models S \cdot (p(q), h) : A$ then there are ρ and v such that $\rho \in S$, $h(\rho)(p) = v$, and v is a function of the form $\lambda(x:B)b$ with $env(H), x:B \vdash b :^e A$.*

Combining Theorem 1 and Proposition 1 we may conclude that such run-time errors never arise in any intermediate configuration reachable from an initial well-typed configuration. Implicitly, this amounts to asserting the safety of

region-based memory management, that defunct regions make no difference to the behaviour of a well-typed configuration. Our π -calculus semantics of regions makes this explicit: we show equationally that direct deletion of defunct regions makes no difference to the semantics of a configuration.

3 A π -Calculus with Groups

In this section, we define a typed π -calculus with groups. In the next, we explain a semantics of our region calculus in this π -calculus. Exactly as in the ambient calculus with groups [5], each name x has a type that includes its group G , and groups may be generated dynamically by a new-group construct, $(\nu G)P$. So as to model the type and effect system of the region calculus, we equip our π -calculus with a novel group-based effect system. In other work [6], not concerned with the region calculus, we consider a simpler version of this π -calculus, with groups but without an effect system, and show that new-group helps keep names secret, in a certain formal sense.

3.1 Syntax

The following table gives the syntax of processes, P . The syntax depends on a set of atomic names, x, y, z, p, q , and a set of groups, G, H . For convenience, we assume that the sets of names and groups are identical to the sets of names and region names, respectively, of the region calculus. We impose a standard constraint [9,14], usually known as locality, that received names may be used for output but not for input. This constraint is actually unnecessary for any of the results of this paper, but is needed for proofs of additional results in the long version [8]. Except for the addition of type annotations and the new-group construct, and the locality constraint, the following syntax and semantics are the same as for the polyadic, choice-free, asynchronous π -calculus [15].

Expressions and Processes:

x, y, p, q	name: variable, channel
$P, Q, R ::=$	process
$x(y_1:T_1, \dots, y_n:T_n).P$	input (no $y_i \in \text{inp}(P)$)
$\bar{x}\langle y_1, \dots, y_n \rangle$	output
$(\nu G)P$	new-group: group restriction
$(\nu x:T)P$	new-name: name restriction
$P \mid Q$	composition
$!P$	replication
0	inactivity

The set $\text{inp}(P)$ contains each name x such that an input process $x(y_1:T_1, \dots, y_n:T_n).P'$ occurs as a subprocess of P , with x not bound. We explain the types T below. In a process $x(y_1:T_1, \dots, y_n:T_n).P$, the names $y_1, \dots, y_n$ are bound; their scope is P . In a group restriction $(\nu G)P$, the group G is bound; its scope

is P . In a name restriction $(\nu x:T)P$, the name x is bound; its scope is P . We identify processes up to the consistent renaming of bound groups and names. We let $fn(P)$ and $fg(P)$ be the sets of free names and free groups, respectively, of a process P . We write $P\{x \leftarrow y\}$ for the outcome of a capture-avoiding substitution of the name y for each free occurrence of the name x in the process P .

Next, we explain the semantics of the calculus informally, by example. We omit type annotations and groups; we shall explain these later.

A process represents a particular state in a π -calculus computation. A state may reduce to a successor when two subprocesses interact by exchanging a tuple of names on a shared communication channel, itself identified by a name. For example, consider the following process:

$$f(x, k').\bar{k'}\langle x \rangle \mid g(y, k').\bar{f}\langle y, k' \rangle \mid \bar{g}\langle 5, k \rangle$$

This is the parallel composition (denoted by the $\mid$ operator) of two input processes $g(y, k').\bar{f}\langle y, k' \rangle$ and $f(x, k').\bar{k'}\langle x \rangle$, and an output process $\bar{g}\langle 5, k \rangle$. The whole process performs two reductions. The first is to exchange the tuple $\langle 5, k \rangle$ on the channel g . The names 5 and k are bound to the input names y and k , leaving $f(x, k').\bar{k'}\langle x \rangle \mid \bar{f}\langle 5, k \rangle$ as the next state. This state itself may reduce to the final state $\bar{k}\langle 5 \rangle$ via an exchange of $\langle 5, k \rangle$ on the channel f .

The process above illustrates how functions may be encoded as processes. Specifically, it is a simple encoding of the example ex_1 from Section 2.1. The input processes correspond to λ -abstractions at addresses f and g ; the output processes correspond to function applications; the name k is a continuation for the whole expression. The reductions described above represent the semantics of the expression: a short internal computation returning the result 5 on the continuation k .

The following is a more accurate encoding:

$$(\nu f)(\nu g)(\overbrace{!f(x, k').\bar{k'}\langle x \rangle}^{f \mapsto \lambda(x)x} \mid \overbrace{!g(y, k').\bar{f}\langle y, k' \rangle}^{g \mapsto \lambda(y)f(y)} \mid \overbrace{\bar{g}\langle 5, k \rangle}^{g(5)})$$

A replication $!P$ is like an infinite parallel array of replicas of P ; we replicate the inputs above so that they may be invoked arbitrarily often. A name restriction $(\nu x)P$ invents a fresh name x with scope P ; we restrict the addresses f and g above to indicate that they are dynamically generated, rather than being global constants.

The other π -calculus constructs are group restriction and inactivity. Group restriction $(\nu G)P$ invents a fresh group G with scope P ; it is the analogue of name restriction for groups. Finally, the $\mathbf{0}$ process represents inactivity.

3.2 Dynamic Semantics

We formalize the semantics of our π -calculus using standard techniques. A reduction relation, $P \rightarrow Q$, means that P evolves in one step to Q . It is defined in terms of an auxiliary structural congruence relation, $P \equiv Q$, that identifies processes we never wish to tell apart.

Structural Congruence: $P \equiv Q$

$P \equiv P$	(Struct Refl)
$Q \equiv P \Rightarrow P \equiv Q$	(Struct Symm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(Struct Trans)
$P \equiv Q \Rightarrow x(y_1:T_1, \dots, y_n:T_n).P \equiv x(y_1:T_1, \dots, y_n:T_n).Q$	(Struct Input)
$P \equiv Q \Rightarrow (\nu G)P \equiv (\nu G)Q$	(Struct GRes)
$P \equiv Q \Rightarrow (\nu x:T)P \equiv (\nu x:T)Q$	(Struct Res)
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(Struct Par)
$P \equiv Q \Rightarrow !P \equiv !Q$	(Struct Repl)
$P \mid \mathbf{0} \equiv P$	(Struct Par Zero)
$P \mid Q \equiv Q \mid P$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$!P \equiv P \mid !P$	(Struct Repl Par)
$x_1 \neq x_2 \Rightarrow (\nu x_1:T_1)(\nu x_2:T_2)P \equiv (\nu x_2:T_2)(\nu x_1:T_1)P$	(Struct Res Res)
$x \notin \text{fn}(P) \Rightarrow (\nu x:T)(P \mid Q) \equiv P \mid (\nu x:T)Q$	(Struct Res Par)
$(\nu G_1)(\nu G_2)P \equiv (\nu G_2)(\nu G_1)P$	(Struct GRes GRes)
$G \notin \text{fg}(T) \Rightarrow (\nu G)(\nu x:T)P \equiv (\nu x:T)(\nu G)P$	(Struct GRes Res)
$G \notin \text{fg}(P) \Rightarrow (\nu G)(P \mid Q) \equiv P \mid (\nu G)Q$	(Struct GRes Par)

Reduction: $P \rightarrow Q$

$\bar{x}\langle y_1, \dots, y_n \rangle \mid x(z_1:T_1, \dots, z_n:T_n).P \rightarrow P\{z_1 \leftarrow y_1\} \dots \{z_n \leftarrow y_n\}$	(Red Interact)
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(Red Par)
$P \rightarrow Q \Rightarrow (\nu G)P \rightarrow (\nu G)Q$	(Red GRes)
$P \rightarrow Q \Rightarrow (\nu x:T)P \rightarrow (\nu x:T)Q$	(Red Res)
$P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$	(Red $\equiv$)

Groups help to type-check names statically but have no dynamic behaviour; groups are not themselves values. The following proposition demonstrates this precisely; it asserts that the reduction behaviour of a typed process is equivalent to the reduction behaviour of the untyped process obtained by erasing all type and group annotations.

Erasing type annotations and group restrictions:

$\text{erase}((\nu G)P) \triangleq \text{erase}(P)$
$\text{erase}((\nu x:T)P) \triangleq (\nu x)\text{erase}(P)$
$\text{erase}(\mathbf{0}) \triangleq \mathbf{0}$
$\text{erase}(P \mid Q) \triangleq \text{erase}(P) \mid \text{erase}(Q)$
$\text{erase}(!P) \triangleq !\text{erase}(P)$
$\text{erase}(x(y_1:T_1, \dots, y_n:T_n).P) \triangleq x(y_1, \dots, y_n).\text{erase}(P)$
$\text{erase}(\bar{x}\langle y_1, \dots, y_n \rangle) \triangleq \bar{x}\langle y_1, \dots, y_n \rangle$

Proposition 2 (Erasure). *For all typed processes P and Q , if $P \rightarrow Q$ then $\text{erase}(P) \rightarrow \text{erase}(Q)$ and if $\text{erase}(P) \rightarrow R$ then there is a typed process Q such that $P \rightarrow Q$ and $R \equiv \text{erase}(Q)$.*

3.3 Static Semantics

The main judgment $E \vdash P : \{G_1, \dots, G_n\}$ of the effect system for the π -calculus means that the process P uses names according to their types and that all its external reads and writes are on channels in groups $G_1, \dots, G_n$. A channel type takes the form $G[T_1, \dots, T_n] \backslash \mathbf{H}$. This stipulates that the name is in group G and that it is a channel for the exchange of n -tuples of names with types $T_1, \dots, T_n$. The set of group names $\mathbf{H}$ is the *hidden effect* of the channel. In the common case when $\mathbf{H} = \emptyset$, we abbreviate the type to $G[T_1, \dots, T_n]$.

As examples of groups, in our encoding of the region calculus we have groups Lit and K for literals and continuations, respectively, and each region ρ is a group. Names of type $\text{Lit}[]$ are in group Lit and exchange empty tuples, and names of type $K[\text{Lit}[]]$ are in group K and exchange names of type $\text{Lit}[]$. In our running example, we have $5 : \text{Lit}[]$ and $k : K[\text{Lit}[]]$. A pointer to a function in a region ρ is a name in group ρ . In our example, we could have $f : \rho'[\text{Lit}[], K[\text{Lit}[]]]$ and $g : \rho[\text{Lit}[], K[\text{Lit}[]]]$.

Given these typings for names, we have $g(y, k').\bar{f}\langle y, k' \rangle : \{\rho, \rho'\}$ because the reads and writes of the process are on the channels g and f whose groups are ρ and ρ' . Similarly, we have $f(x, k').\bar{k}'\langle x \rangle : \{\rho', K\}$ and $\bar{g}\langle 5, k \rangle : \{\rho\}$. The composition of these three processes has effect $\{\rho, \rho', K\}$, the union of the individual effects.

The idea motivating hidden effects is that an input process listening on a channel may represent a passive resource (for example, a function) that is only invoked if there is an output on the channel. The hidden effect of a channel is an effect that is masked in an input process, but incurred by an output process. In the context of our example, our formal translation makes the following type assignments: $f : \rho'[\text{Lit}[], K[\text{Lit}[]]] \backslash \{K\}$ and $g : \rho[\text{Lit}[], K[\text{Lit}[]]] \backslash \{K, \rho'\}$. We then have $f(x, k').\bar{k}'\langle x \rangle : \{\rho'\}$, $g(y, k').\bar{f}\langle y, k' \rangle : \{\rho\}$, and $\bar{g}\langle 5, k \rangle : \{\rho, \rho', K\}$. The hidden effects are transferred from the function bodies to the process $\bar{g}\langle 5, k \rangle$ that invokes the functions. This transfer is essential in the proof of our main garbage collection result, Theorem 5.

The effect of a replicated or name-restricted process is the same as the original process. For example, abbreviating the types for f and g , we have: $(\nu f:\rho')(\nu g:\rho)(!f(x, k').\bar{k}'\langle x \rangle \mid !g(y, k').\bar{f}\langle y, k' \rangle \mid \bar{g}\langle 5, k \rangle) : \{\rho, \rho', K\}$.

On the other hand, the effect of a group-restriction $(\nu G)P$ is the same as that of P , except that G is deleted. This is because there can be no names free in P of group G ; any names of group G in P must be internally introduced by name-restrictions. Therefore, $(\nu G)P$ has no external reads or writes on G channels. For example, $(\nu \rho')(\nu f)(\nu g)(!f(x, k').\bar{k}'\langle x \rangle \mid !g(y, k').\bar{f}\langle y, k' \rangle \mid \bar{g}\langle 5, k \rangle) : \{\rho, K\}$.

The following tables describe the syntax of types and environments, the judgments and the rules defining our effect system. Let $fg(G[T_1, \dots, T_n] \backslash \mathbf{H}) \triangleq \{G\} \cup fg(T_1) \cup \dots \cup fg(T_n) \cup \mathbf{H}$.

Syntax of Types and Environments, Typing Judgments:

$\mathbf{G}, \mathbf{H} ::= \{G_1, \dots, G_k\}$	finite set of name groups
$T ::= G[T_1, \dots, T_n] \setminus \mathbf{H}$	type of channel in group G with hidden effect $\mathbf{H}$
$E ::= \emptyset \mid E, G \mid E, x:T$	environment
$E \vdash \diamond$	good environment
$E \vdash T$	good channel type T
$E \vdash x : T$	good name x of channel type T
$E \vdash P : \mathbf{H}$	good process P with effect $\mathbf{H}$

Typing Rules:

(Env $\emptyset$)	(Env x)	(Env G)
$\emptyset \vdash \diamond$	$\frac{E \vdash T \quad x \notin \text{dom}(E)}{E, x:T \vdash \diamond}$	$\frac{E \vdash \diamond \quad G \notin \text{dom}(E)}{E, G \vdash \diamond}$
(Type Chan)	(Exp x)	
$\frac{E \vdash \diamond \quad \{G\} \cup \mathbf{H} \subseteq \text{dom}(E) \quad E \vdash T_1 \quad \dots \quad E \vdash T_n}{E \vdash G[T_1, \dots, T_n] \setminus \mathbf{H}}$	$\frac{E', x:T, E'' \vdash \diamond}{E', x:T, E'' \vdash x : T}$	
(Proc Input)	$\frac{E \vdash x : G[T_1, \dots, T_n] \setminus \mathbf{H} \quad E, y_1:T_1, \dots, y_n:T_n \vdash P : \mathbf{G}}{E \vdash x(y_1:T_1, \dots, y_n:T_n).P : \{G\} \cup (\mathbf{G} - \mathbf{H})}$	
(Proc Output)	$\frac{E \vdash x : G[T_1, \dots, T_n] \setminus \mathbf{H} \quad E \vdash y_1 : T_1 \quad \dots \quad E \vdash y_n : T_n}{E \vdash \bar{x}\langle y_1, \dots, y_n \rangle : \{G\} \cup \mathbf{H}}$	
(Proc GRes)	(Proc Res)	(Proc Par)
$\frac{E, G \vdash P : \mathbf{H}}{E \vdash (\nu G)P : \mathbf{H} - \{G\}}$	$\frac{E, x:T \vdash P : \mathbf{H}}{E \vdash (\nu x:T)P : \mathbf{H}}$	$\frac{E \vdash P : \mathbf{G} \quad E \vdash Q : \mathbf{H}}{E \vdash P \mid Q : \mathbf{G} \cup \mathbf{H}}$
(Proc Repl)	(Proc Zero)	(Proc Subsum)
$\frac{E \vdash P : \mathbf{H}}{E \vdash !P : \mathbf{H}}$	$\frac{E \vdash \diamond}{E \vdash \mathbf{0} : \emptyset}$	$\frac{E \vdash P : \mathbf{G} \quad \mathbf{G} \subseteq \mathbf{H} \subseteq \text{dom}(E)}{E \vdash P : \mathbf{H}}$

The rules for good environments and good channel types ensure that declared names and groups are distinct, and that all the names and groups occurring in a type are declared. The rules for good processes ensure that names are used for input and output according to their types, and compute an effect that includes the groups of all the free names used for input and output.

In the special case when the hidden effect $\mathbf{H}$ is $\emptyset$, (Proc Input) and (Proc Output) specialise to the following:

$\frac{E \vdash x : G[T_1, \dots, T_n] \setminus \emptyset \quad E, y_1:T_1, \dots, y_n:T_n \vdash P : \mathbf{G}}{E \vdash x(y_1:T_1, \dots, y_n:T_n).P : \{G\} \cup \mathbf{G}}$	$\frac{E \vdash x : G[T_1, \dots, T_n] \setminus \emptyset \quad E \vdash y_1 : T_1 \quad \dots \quad E \vdash y_n : T_n}{E \vdash \bar{x}\langle y_1, \dots, y_n \rangle : \{G\}}$

In this situation, we attribute all the effect $\mathbf{G}$ of the prefixed process P to the input process $x(y_1:T_1, \dots, y_n:T_n).P$. The effect $\mathbf{G}$ of P is entirely excluded from the hidden effect, since $\mathbf{H} = \emptyset$.

A dual special case is when the effect of the prefixed process P is entirely included in the hidden effect $\mathbf{H}$. In this case, (Proc Input) and (Proc Output) specialise to the following:

$$\frac{E \vdash x : G[T_1, \dots, T_n] \setminus \mathbf{H} \quad E, y_1:T_1, \dots, y_n:T_n \vdash P : \mathbf{H}}{E \vdash x(y_1:T_1, \dots, y_n:T_n).P : \{G\}} \quad \frac{E \vdash x : G[T_1, \dots, T_n] \setminus \mathbf{H} \quad E \vdash y_1 : T_1 \quad \dots \quad E \vdash y_n : T_n}{E \vdash \bar{x}\langle y_1, \dots, y_n \rangle : \{G\} \cup \mathbf{H}}$$

The effect of P is not attributed to the input $x(y_1:T_1, \dots, y_n:T_n).P$ but instead is transferred to any outputs in the same group as x . If there are no such outputs, the process P will remain blocked, so it is safe to discard its effects.

These two special cases of (Proc Input) and (Proc Output) are in fact sufficient for the encoding of the region calculus presented in Section 4; we need the first special case for typing channels representing continuations, and the second special case for typing channels representing function pointers. For simplicity, our actual rules (Proc Input) and (Proc Output) combine both special cases; an alternative would be to have two different kinds of channel types corresponding to the two special cases.

The rule (Proc GRes) discards G from the effect of a new-group process $(\nu G)P$, since, in P , there can be no free names of group G (though there may be restricted names of group G). The rule (Proc Subsum) is a rule of effect subsumption. We need this rule to model the effect subsumption in rule (Exp Fun) of the region calculus. The other rules for good processes simply compute the effect of a whole process in terms of the effects of its parts.

We can prove a standard subject reduction result.

Proposition 3. *If $E \vdash P : \mathbf{H}$ and $P \rightarrow Q$ then $E \vdash Q : \mathbf{H}$.*

Next, a standard definition of the barbs exhibited by a process formalizes the idea of the external reads and writes through which a process may interact with its environment. Let a *barb*, β , be either a name x or a co-name $\bar{x}$.

Exhibition of a barb: $P \downarrow \beta$

$$\frac{}{x(y_1:T_1, \dots, y_n:T_n).P \downarrow x} \quad \frac{}{\bar{x}\langle y_1, \dots, y_n \rangle \downarrow \bar{x}}$$

$$\frac{P \downarrow \beta}{(\nu G)P \downarrow \beta} \quad \frac{P \downarrow \beta \quad \beta \notin \{x, \bar{x}\}}{(\nu x:T)P \downarrow \beta} \quad \frac{P \downarrow \beta}{P \mid Q \downarrow \beta} \quad \frac{P \equiv Q \quad Q \downarrow \beta}{P \downarrow \beta}$$

The following asserts the soundness of the effect system. The group of any barb of a process is included in its effect.

Proposition 4. *If $E \vdash P : \mathbf{H}$ and $P \downarrow \beta$ with $\beta \in \{x, \bar{x}\}$ then there is a type $G[T_1, \dots, T_n] \setminus \mathbf{G}$ such that $E \vdash x : G[T_1, \dots, T_n] \setminus \mathbf{G}$ and $G \in \mathbf{H}$.*

4 Encoding Regions as Groups

This section interprets the region calculus in terms of our π -calculus. Most of the ideas of the translation are standard, and have already been illustrated by example. A function value in the heap is represented by a replicated input process, awaiting the argument and a continuation on which to return a result. A function is invoked by sending it an argument and a continuation. Region names and *letregion* ρ are translated to groups and $(\nu\rho)$, respectively.

The remaining construct of our region calculus is sequencing: *let* $x = a$ *in* b . Assuming a continuation k , we translate this to $(\nu k')(\llbracket a \rrbracket k' \mid k'(x).\llbracket b \rrbracket k)$. This process invents a fresh, intermediate continuation k' . The process $\llbracket a \rrbracket k'$ evaluates a returning a result on k' . The process $k'(x).\llbracket b \rrbracket k$ blocks until the result x is returned on k' , then evaluates b , returning its result on k .

The following tables interpret the types, environments, expressions, regions, and configurations of the region calculus in the π -calculus. In particular, if $S \cdot (a, h)$ is a configuration, then $\llbracket S \cdot (a, h) \rrbracket k$ is its translation, a process that returns any eventual result on the continuation k . In typing the translation, we assume two global groups: a group, K , of continuations and a group, Lit , of literals. The environment $\llbracket \emptyset \rrbracket$ declares these groups and also a typing $\ell_i:Lit$ for each of the literals $\ell_1, \dots, \ell_n$.

Translating of the region calculus to the π -calculus:

$\llbracket A \rrbracket$	type modelling the type A
$\llbracket E \rrbracket$	environment modelling environment E
$\llbracket a \rrbracket k$	process modelling term a , answer on k
$\llbracket p \mapsto v \rrbracket$	process modelling value v at pointer p
$\llbracket r \rrbracket$	process modelling region r
$\llbracket S \cdot (a, h) \rrbracket k$	process modelling configuration $S \cdot (a, h)$

In the following equations, where necessary to construct type annotations in the π -calculus, we have added type subscripts to the syntax of the region calculus. The notation $\prod_{i \in I} P_i$ for some finite indexing set $I = \{i_1, \dots, i_n\}$ is short for the composition $P_{i_1} \mid \dots \mid P_{i_n} \mid \mathbf{0}$.

Translation rules:

$\llbracket Lit \rrbracket \triangleq Lit[]$
$\llbracket (A \xrightarrow{e} B) \text{ at } \rho \rrbracket \triangleq \rho[\llbracket A \rrbracket, K[\llbracket B \rrbracket]] \setminus (e \cup \{K\})$
$\llbracket \emptyset \rrbracket \triangleq K, Lit, \ell_1:Lit[], \dots, \ell_n:Lit[]$
$\llbracket E, \rho \rrbracket \triangleq \llbracket E \rrbracket, \rho$
$\llbracket E, x:A \rrbracket \triangleq \llbracket E \rrbracket, x:\llbracket A \rrbracket$
$\llbracket x \rrbracket k \triangleq \bar{k}\langle x \rangle$
$\llbracket \text{let } x = a_A \text{ in } b \rrbracket k \triangleq (\nu k':K[\llbracket A \rrbracket])(\llbracket a \rrbracket k' \mid k'(x:\llbracket A \rrbracket).\llbracket b \rrbracket k)$
$\llbracket p(q) \rrbracket k \triangleq \bar{p}\langle q, k \rangle$

$$\begin{aligned}
\llbracket \text{letregion } \rho \text{ in } a \rrbracket k &\triangleq (\nu \rho) \llbracket a \rrbracket k \\
\llbracket (v \text{ at } \rho)_A \rrbracket k &\triangleq (\nu p: \llbracket A \rrbracket) (\llbracket p \mapsto v \rrbracket \mid \bar{k} \langle p \rangle) \\
\llbracket p \mapsto \lambda(x:A) b_B \rrbracket &\triangleq !p(x: \llbracket A \rrbracket, k: K[\llbracket B \rrbracket]). \llbracket b \rrbracket k \\
\llbracket (p_i \mapsto v_i)_{i \in 1..n} \rrbracket &\triangleq \prod_{i \in 1..n} \llbracket p_i \mapsto v_i \rrbracket \\
\llbracket (\rho_i \mapsto r_i)_{i \in 1..n} \rrbracket &\triangleq \prod_{i \in 1..n} \llbracket r_i \rrbracket \\
\llbracket S \cdot (a, h_H) \rrbracket k &\triangleq (\nu \rho_{\text{defunct}}) (\nu \llbracket ptr(H) \rrbracket) (\llbracket a \rrbracket k \mid \llbracket h \rrbracket) \text{ if } \{\rho_{\text{defunct}}\} = \text{dom}(H) - S
\end{aligned}$$

The following theorem asserts that the translation preserves the static semantics of the region calculus.

Theorem 2 (Static Adequacy).

- (1) If $E \vdash \diamond$ then $\llbracket E \rrbracket \vdash \diamond$.
- (2) If $E \vdash A$ then $\llbracket E \rrbracket \vdash \llbracket A \rrbracket$.
- (3) If $E \vdash a :^e A$ and $k \notin \text{dom}(\llbracket E \rrbracket)$ then $\llbracket E \rrbracket, k:K[\llbracket A \rrbracket] \vdash \llbracket a \rrbracket k : e \cup \{K\}$.
- (4) If $H \models h$ and $\rho \in \text{dom}(H)$ then $\llbracket \text{env}(H) \rrbracket \vdash \llbracket h(\rho) \rrbracket : \{\rho\}$.
- (5) If $H \models S \cdot (a, h) : A$ and $k \notin \llbracket \text{env}(H) \rrbracket$ then $\llbracket \text{env}(H) \rrbracket, k:K[\llbracket A \rrbracket] \vdash \llbracket a \rrbracket k \mid \llbracket h \rrbracket : \text{dom}(H) \cup \{K\}$ and also $\llbracket \emptyset \rrbracket, S, k:K[\llbracket A \rrbracket] \vdash \llbracket S \cdot (a, h) \rrbracket k : S \cup \{K\}$.

Next we state that the translation preserves the dynamic semantics. First, we take our process equivalence to be barbed congruence [16], a standard operational equivalence for the π -calculus. We use a typed version of (weak) barbed congruence, as defined by Pierce and Sangiorgi [19]; the long version of this paper contains the detailed definition. Then, our theorem states that if one region calculus configuration evaluates to another, their π -calculus interpretations are equivalent. In the following, let $E \vdash P$ mean there is an effect $\mathbf{G}$ such that $E \vdash P : \mathbf{G}$.

Typed process equivalence: $E \vdash P \approx Q$

For all typed processes P and Q , let $E \vdash P \approx Q$ mean that $E \vdash P$ and $E \vdash Q$ and that P and Q are barbed congruent.

Theorem 3 (Dynamic Adequacy). *If $H \models S \cdot (a, h) : A$ and $S \cdot (a, h) \Downarrow (p', h')$ then there is H' such that $H \asymp H'$ and $H + H' \models S \cdot (p', h') : A$ and for all $k \notin \text{dom}_2(H + H') \cup L$, $\llbracket \emptyset \rrbracket, S, k:K[\llbracket A \rrbracket] \vdash \llbracket S \cdot (a, h) \rrbracket k \approx \llbracket S \cdot (p', h') \rrbracket k$.*

Recall the evaluations of the examples ex_1 and ex_2 given previously. From Theorem 3 we obtain the following equations (in which we abbreviate environments and types for the sake of clarity):

$$\begin{aligned}
\llbracket \{\rho\} \cdot (ex_1, h) \rrbracket k &\approx (\nu \rho') (\nu f:p') (\nu g:\rho) (\llbracket f \mapsto \lambda(x)x \rrbracket \mid \llbracket g \mapsto \lambda(y)f(y) \rrbracket \mid \bar{k} \langle 5 \rangle) \\
\llbracket \{\rho\} \cdot (ex_2, h) \rrbracket k &\approx (\nu \rho') (\nu f:p') (\nu g:\rho) (\nu j:\rho) \\
&\quad (\llbracket f \mapsto \lambda(x)x \rrbracket \mid \llbracket g \mapsto \lambda(f)5 \rrbracket \mid \llbracket j \mapsto \lambda(z)g(f) \rrbracket \mid \bar{k} \langle j \rangle)
\end{aligned}$$

Next, we present a general π -calculus theorem that has as a corollary a theorem asserting that defunct regions may be deleted without affecting the meaning of a configuration.

Suppose there are processes P and R such that R has effect $\{G\}$ but G is not in the effect of P . So R only interacts on names in group G , but P never interacts on names in group G , and therefore there can be no interaction between P and R . Moreover, if P and R are the only sources of inputs or outputs in the scope of G , then R has no external interactions, and therefore makes no difference to the behaviour of the whole process. The following makes this idea precise equationally. We state the theorem in terms of the notation $(\nu E)P$ defined by the equations: $(\nu \emptyset)P \triangleq P$, $(\nu E, x:T)P \triangleq (\nu E)(\nu x:T)P$, and $(\nu E, G)P \triangleq (\nu E)(\nu G)P$. The proof proceeds by constructing a suitable bisimulation relation.

Theorem 4. *If $E, G, E' \vdash P : \mathbf{H}$ and $E, G, E' \vdash R : \{G\}$ with $G \notin \mathbf{H}$, then $E \vdash (\nu G)(\nu E')(P \mid R) \approx (\nu G)(\nu E')P$.*

Now, by applying this theorem, we can delete the defunct region ρ' from our two examples. We obtain:

$$\begin{aligned} & (\nu \rho')(\nu f:\rho')(\nu g:\rho)(\llbracket f \mapsto \lambda(x)x \rrbracket \mid \llbracket g \mapsto \lambda(y)f(y) \rrbracket \mid \bar{k}\langle 5 \rangle) \\ & \approx (\nu \rho')(\nu f:\rho')(\nu g:\rho)(\llbracket g \mapsto \lambda(y)f(y) \rrbracket \mid \bar{k}\langle 5 \rangle) \\ & (\nu \rho')(\nu f:\rho')(\nu g:\rho)(\nu j:\rho)(\llbracket f \mapsto \lambda(x)x \rrbracket \mid \llbracket g \mapsto \lambda(f)5 \rrbracket \mid \llbracket j \mapsto \lambda(z)g(f) \rrbracket \mid \bar{k}\langle j \rangle) \\ & \approx (\nu \rho')(\nu f:\rho')(\nu g:\rho)(\nu j:\rho)(\llbracket g \mapsto \lambda(f)5 \rrbracket \mid \llbracket j \mapsto \lambda(z)g(f) \rrbracket \mid \bar{k}\langle j \rangle) \end{aligned}$$

The first equation illustrates the need for hidden effects. The hidden effect of g is $\{K, \rho'\}$, and so the overall effect of the process $\llbracket g \mapsto \lambda(y)f(y) \rrbracket \mid \bar{k}\langle 5 \rangle$ is simply $\{\rho, K\}$. This effect does not contain ρ' and so the theorem justifies deletion of the process $\llbracket f \mapsto \lambda(x)x \rrbracket$, whose effect is $\{\rho'\}$. In an effect system for the π -calculus without hidden effects, the effect of $\llbracket g \mapsto \lambda(y)f(y) \rrbracket \mid \bar{k}\langle 5 \rangle$ would include ρ' , and so the theorem would not be applicable.

A standard garbage collection principle in the π -calculus is that if f does not occur free in P , then $(\nu f)(!f(x, k).R \mid P) \approx P$. One might hope that this principle alone would justify de-allocation of defunct regions. But neither of our example equations is justified by this principle; in both cases, the name f occurs in the remainder of the process. We need an effect system to determine that f is not actually invoked by the remainder of the process.

The two equations displayed above are instances of our final theorem, a corollary of Theorem 4. It asserts that deleting defunct regions makes no difference to the behaviour of a configuration:

Theorem 5. *Suppose $H \models S \cdot (a, h) : A$ and $k \notin \text{dom}_2(H) \cup L$. Let $\{\rho_{\text{defunct}}\} = \text{dom}(H) - S$. Then we can derive the equation $\llbracket \emptyset \rrbracket, S, k:K \llbracket A \rrbracket \vdash \llbracket S \cdot (a, h) \rrbracket k \approx (\nu \rho_{\text{defunct}})(\nu \llbracket \text{ptr}(H) \rrbracket)(\llbracket a \rrbracket k \mid \prod_{\rho \in S} \llbracket H(\rho) \rrbracket)$.*

5 Conclusions

We showed that the static and dynamic semantics of Tofte and Talpin's region calculus are preserved by a translation into a typed π -calculus. The *letregion*

construct is modelled by a new-group construct originally introduced into process calculi in the setting of the ambient calculus [5]. We showed that the rather subtle correctness of memory de-allocation in the region calculus is an instance of Theorem 4, a new garbage collection principle for the π -calculus. The translation is an example of how the new-group construct accounts for the type generativity introduced by *letregion*, just as the standard new-name construct of the π -calculus accounts for dynamic generation of values.

Banerjee, Heintze, and Riecke [3] give an alternative proof of the soundness of region-based memory management. Theirs is obtained by interpreting the region calculus in a polymorphic λ -calculus equipped with a new binary type constructor $\#$ that behaves like a union or intersection type. Their techniques are those of denotational semantics, completely different from the operational techniques of this paper. The formal connections between the two approaches are not obvious but would be intriguing to investigate. A possible advantage of our semantics in the π -calculus is that it could easily be extended to interpret a region calculus with concurrency, but that remains future work. Another line of future work is to consider the semantics of other region calculi [2,7,11] in terms of the π -calculus. Finally, various researchers [18,23] have noted a connection between the monadic encapsulation of state in Haskell [12] and regions; hence it would be illuminating to interpret monadic encapsulation in the π -calculus.

Acknowledgements. Luca Cardelli participated in the initial discussions that led to this paper. We had useful conversations with Cédric Fournet, Giorgio Ghelli and Mads Tofte. Luca Cardelli, Tony Hoare, and Andy Moran commented on a draft.

References

1. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Information and Computation*, 148:1–70, 1999. An extended version appears as Research Report 149, Digital Equipment Corporation Systems Research Center, January 1998.
2. A. Aiken, M. Fähndrich, and R. Levien. Better static memory management: Improving region-based analysis of higher-order languages. In *Proceedings PLDI'95*, pages 174–185, 1995.
3. A. Banerjee, N. Heintze, and J. Riecke. Region analysis and the polymorphic lambda calculus. In *Proceedings LICS'99*, 1999.
4. L. Birkedal, M. Tofte, and M. Vejlsstrup. From region inference to von Neumann machines via region representation inference. In *Proceedings POPL'96*, pages 171–183. 1996.
5. L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *Proceedings TCS2000*, Lecture Notes in Computer Science. Springer, 2000. To appear.
6. L. Cardelli, G. Ghelli, and A. D. Gordon. Group creation and secrecy. In *Proceedings Concur'00*, Lecture Notes in Computer Science. Springer, 2000. To appear.
7. K. Crary, D. Walker, and G. Morrisett. Typed memory management in a calculus of capabilities. In *Proceedings POPL'99*, pages 262–275, 1999.

8. S. Dal Zilio and A. D. Gordon. Region analysis and a π -calculus with groups. Technical Report MSR-TR-2000-57, Microsoft Research, 2000.
9. C. Fournet and G. Gonthier. The reflexive CHAM and the Join-calculus. In *Proceedings POPL'96*, pages 372–385, 1996.
10. D. K. Gifford and J. M. Lucassen. Integrating functional and imperative programming. In *Proceedings L&FP'86*, pages 28–38, 1986.
11. J. Hughes and L. Pareto. Recursion and dynamic data-structures in bounded space: Towards embedded ML programming. In *Proceedings ICFP'99*, pages 70–81, 1999.
12. J. Launchbury and S. Peyton Jones. State in Haskell. *Lisp and Symbolic Computation*, 8(4):293–341, 1995.
13. X. Leroy. A syntactic theory of type generativity and sharing. *Journal of Functional Programming*, 6(5):667–698, 1996.
14. M. Merro and D. Sangiorgi. On asynchrony in name-passing calculi. In *Proceedings ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 856–867. Springer, 1998.
15. R. Milner. *Communicating and Mobile Systems: the π -Calculus*. Cambridge University Press, 1999.
16. R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proceedings ICALP'92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.
17. R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.
18. E. Moggi and F. Palumbo. Monadic encapsulation of effects: a revised approach. In *Proceedings HOOTS99*, volume 26 of *Electronic Notes in Theoretical Computer Science*, pages 119–136. Elsevier, 1999.
19. B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.
20. B. C. Pierce and D. 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*, G. Plotkin, C. Stirling, and M. Tofte, editors, MIT Press, 2000.
21. C. V. Russo. Standard ML type generativity as existential quantification. Technical Report ECS-LFCS-96-344, LFCS, University of Edinburgh, 1996.
22. D. Sangiorgi. Interpreting functions as π -calculus processes: a tutorial. Technical Report 3470, INRIA, 1998. Draft chapter to appear in *The pi-calculus: a theory of mobile processes*, D. Sangiorgi and W. Walker, Cambridge University Press, 2000.
23. M. Semmelroth and A. Sabry. Monadic encapsulation in ML. In *Proceedings ICFP'99*, pages 8–17, 1999.
24. J.-P. Talpin and P. Jouvelot. Polymorphic type, region and effect inference. *Journal of Functional Programming*, 2(3):245–271, 1992.
25. C. J. Taylor. *Formalising and Reasoning about Fudgets*. PhD thesis, University of Nottingham, 1998. Available as Technical Report NOTTCS-TR-98-4.
26. M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.
27. P. Wadler. The marriage of effects and monads. In *Proceedings ICFP'98*, pages 63–74, 1998.
28. D. Walker. Objects in the pi-calculus. *Information and Computation*, 116(2):253–271, 1995.

Abstract Data Types in Computer Algebra

James H. Davenport*

Department of Mathematical Sciences, University of Bath, Bath BA2 7AY, England
`jhd@maths.bath.ac.uk`

Abstract. The theory of abstract data types was developed in the late 1970s and the 1980s by several people, including the “ADJ” group, whose work influenced the design of Axiom. One practical manifestation of this theory was the OBJ-3 system. An area of computing that cries out for this approach is computer algebra, where the objects of discourse are mathematical, generally satisfying various algebraic rules. There have been various theoretical studies of this in the literature: [36,42,45] to name but a few.

The aim of this paper is to report on the practical applications of this theory within computer algebra, and also to outline some of the theoretical issues raised by this practical application. We also give a substantial bibliography.

1 Introduction

The theory of abstract data types has been around since the mid-1970’s [12, 21,22,27,28,29,30,31,55,56]. It has been taken up in computer algebra [8,10,32, 35,40], as a means of structuring some computer algebra systems. Some of the theory of this has been described before [11,49], but not as applied to a general-purpose system. The general basis for this is a belief that “algebra” (in the sense of what computer algebra systems do, or ought to do) can be modelled by the abstract theory of order-sorted algebras, generally initial [27].

However, logical cleanness is only one of the criteria on which an algebra system is judged (and, pragmatically if regrettably, one of the least of the criteria). Other criteria include performance, richness, extensibility and ease-of-use. All of these mean that a “pure” abstract data type system such as OBJ-3 cannot be converted into a reasonable algebra system, and the designers of algebra systems have been forced into various compromises, and also into various extensions of the methodology.

Performance. Order-sorted equational reasoning is not complete (else it could solve the word problem for groups [52]), and can be very slow on non-trivial examples. In the example of rational numbers given in [49] over thirty steps are needed to prove in his model of $\mathbf{Q}$ that $\frac{2}{-1} = \frac{-2}{1}$.

* Much of the paper was written while the author held the Ontario Research Chair in Computer Algebra at the University of Western Ontario.

Richness. An algebra system often needs several implementations of the same mathematical object. One obvious example is the requirement for both dense and sparse matrices. In the world of polynomials, there are many possible representations [53]. Many systems only implement one (or only expose one to the user, more accurately), but for reasons of efficiency we may well want, for example, dense polynomials (especially univariate) as well as sparse. In the world of Gröbner bases [5], the ordering imposed on the monomials is crucial, and indeed the “FGLM” algorithm [23] relies on having polynomials in two different orderings available simultaneously. If we are to allow the user access to these algorithms, we need to expose polynomials with multiple orderings to the user.

Extensibility. The system must enable the user to add new mathematical objects, not just new operations on existing objects. This was the major weakness with the algebra systems of the 1970s and early 1980s: the choice of mathematical objects was essentially fixed by the system designer, and the user could only add new procedures on these objects. However, the user also wants the new objects to operate with the existing ones: e.g. having defined a new data type D , the user may want polynomials over D , or matrices with entries from D , with the same simplicity as polynomials over the integers or matrices with integer coefficients.

The user also wants this extensibility to be error-free. While one can clearly not expect to protect against all bugs in the user’s code, there are two things that one can reasonably expect: that an extension does not break existing code, and that there are mechanisms to ensure that uses of these new data types are mathematically correct, e.g. that if D ’s multiplication is not commutative, polynomials over D will respect this.

Ease-of-use. These systems are designed to be interactive and “natural” to the mathematically trained. Consider the simple input $\mathbf{x}+1$. Conventionally, $\mathbf{x}$ is a symbol, whereas 1 is a (positive) integer, and our system is unlikely to have defined an addition operator with this signature (and indeed, if the system did work this way, extensibility would be a major problem, as we would end up with an $O(n^2)$ problem of defining $+$ between every pair of types). We could force the user to convert both arguments into a type which did support addition (in Axiom we would write

$$(\mathbf{x}::\text{POLY}(\text{INT}))+(1::\text{POLY}(\text{INT}))$$

to achieve this, where $\text{POLY}(\text{INT})$ is itself a built-in abbreviation for $\text{Polynomial}(\text{Integer})$). But this is intolerably clumsy (we have gone from 3 symbols to 29) and error-prone. The usual solution, outlined in [24], is for the system to deduce an appropriate type (a process we will call *resolution*, since it mainly involves the resolution of the types of arguments) and the convert the arguments into the appropriate type (a process generally called *coercion*).

This untyped “ease of use” can have its drawbacks: see the examples in section 6.3.

2 Systems

The two major computer algebra systems based on an Abstract Data Type view are Axiom [39,40] (formerly Scratchpad II) and Magma [8,9], though there have been several other systems such as Views [2,3] and Weyl [60] or extensions, notably the GAUSS¹ extension to Maple [44,32]. The venerable system Reduce [34] also adopted a significant chunk of this methodology [10]. The extension to Reduce defined in [35] has not, however, seen the light of day. The author has worked largely on Axiom, so most examples are taken from there.

The major incentives for taking an Abstract Data Type view when designing a computer algebra system are the following.

1. Economy of implementation. By the late 1970s, Reduce [34] had at least five sets of code for doing linear algebra, depending on the algebraic data types involved and the precise results required (determinants, triangularisation etc.). In fact, at least two different algorithms were involved, but this was a result of quasi-arbitrary choices by the implementors, rather than any tuning, and there was no way of changing algorithm short of re-implementing. Similar problems plagued Macsyma [7], at the time the most comprehensive computer algebra system.
2. Ease of extension. This was a particularly relevant goal for the Reduce extension described by [10], who wanted to be able to add new objects, e.g. algebraic numbers [1], and have the rest of the system operate on them with no additional changes.
3. Uniformity. It should be possible to add two objects by calling `plus` (or possibly `+`) irrespective of what they are, rather than having to call `matrix_plus`, `polynomial_plus`, `fraction_plus` etc. At the user interface level, this is normally considered vital², but is generally provided via a combination of some kind of interpreted type dispatch and a “most general type” (e.g. a fraction of polynomials in Reduce) into which all objects are shoe-horned.
4. Correctness. There are two classical bugs which have manifested themselves in various computer algebra systems over the years. One is over-zealous use of the binomial theorem:

$$(x + y)^2 \rightarrow x^2 + 2xy + y^2$$

is only valid if x and y commute. For example, if $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $B = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$, then $(A + B)^2 = \begin{pmatrix} 6 & 4 \\ 8 & 6 \end{pmatrix}$, but $A^2 + 2AB + B^2 = \begin{pmatrix} 8 & 4 \\ 8 & 4 \end{pmatrix}$ (which is actually singular, unlike the correct answer).

¹ Now renamed as “Domains” for legal reasons, and incorporated as part of Maple.

² Though Maple does not fully provide this feature, and distinguishes commutative multiplication `*` from non-commutative multiplication `&*` — renamed `.` in Maple VI. For reasons described in section 6.3, it also distinguishes between `factor` and `ifactor`.

The other classic bug is over-zealous use of modular arithmetic. Early versions of Reduce [34] implemented arithmetic modulo 17 (say) via

```
on modular; setmod 17;
```

which instructed Reduce to perform modular arithmetic, and to use the modulus 17. This meant, unfortunately, that $x^9 * x^8 \rightarrow x^{17} \rightarrow x^0 \rightarrow 1$. This is clearly incorrect for symbolic x , and even for numeric x over $\text{GF}(17)$, since $x^{17} \rightarrow x$ for $x \in \text{GF}(17)$. This can be “fixed” by arranging for the simplifier to turn off the `modular` flag when handling exponents (easier said than done), but similar fixes need to be made to array index arithmetic etc. The system described in [10] solves this problem by tagging the *data objects* representing the coefficients with the modular property, but not the exponents.

There are also some practical constraints that need to be satisfied.

1. Efficiency of implementation. In particular (nearly) all the type inference work needs to be done before “run-time”, though in fact (with suitable caching) it is possible to do non-trivial work at “instantiation-time” (e.g. the moment when `PrimeField` is applied to 17 to build the data type for $\text{GF}(17)$), which replaces the traditional “link-time”.
2. Efficiency of algorithms. This means that it must be possible to use efficient algorithms where appropriate, e.g. commutative exponentiation should use the binomial method where applicable; determinant calculations should use fraction-free methods [4,15] where appropriate.
3. Economy of implementation. If we have the concept of an Abelian group, i.e. with binary ‘+’ and unary ‘-’, we should not need to have to implement binary ‘-’ every time. However, it may be important to do so occasionally, since implementing $A - B$ as $A + (-B)$ naïvely can double the amount of working storage required — the obvious example is that of matrices.
4. Usability. The typing problem in computer algebra is harder than usual, since literals are heavily overloaded. For example, in `x+2`, the variable `x` is to be interpreted as $x \in \mathbf{Z}[x]$, and the integer 2 as $2 \in \mathbf{Z}[x]$. If this had to be made explicit the system would be unusable. This poses three problems.
 - Type inference/resolution. In the example above, why was $\mathbf{Z}[x]$ the “right” answer? Why not $\mathbf{Q}[x]$, or³ $\mathbf{Z}(x)$, or even $\text{GF}(29)[x]$. The intuitive answer is “minimality”, but the author knows of no general way to formalise this.
 - Type coercion See [24,54] and section 4.
 - Type retraction. Is $12/6 = 2$? On the one hand, clearly yes, but on the other hand, $12/6$ should have the same type as $12/7$, i.e. $\mathbf{Q}$ rather than $\mathbf{Z}$. This matters since, as we will see in section 6.3, the result of some operations can depend on the type as well as the value. It would be nice if, certainly between interactive steps at least, a system could either retract $12/6 = 2$ from $\mathbf{Q}$ to $\mathbf{Z}$, or at least observe that such a

³ This domain is generally written $\mathbf{Q}(x)$, but in fact it is normally implemented as the field of fractions of $\mathbf{Z}[x]$, so, reading “(…)” as “the field of fractions of the polynomial ring of”, $\mathbf{Z}(x)$ is more natural.

retraction was possible. While this is a toy example, observing that a polynomial in $\mathbf{Q}[x, y, z]$ could in fact be in $\mathbf{Z}[x, y, z]$, or in $\mathbf{Q}[x, z]$ is harder. Indeed $\mathbf{Q}[x_1, \dots, x_n]$ has potentially 2^{n+1} retractions, though if the consistency problem were solved one could investigate $n + 1$ initial retractions ($\mathbf{Q} \rightarrow \mathbf{Z}$ and the n variable omissions) and recurse. Again, the author knows of no systematic work on this problem.

3 The Axiom Approach

Axiom [39] takes the following approach. A **category** is a multi-sorted signature together with a set of axioms that the operations from that signature must satisfy. In this respect it is similar to a **magma** [8,9] or a theory [26]. An example would be a group, which in Axiom could⁴ be defined as follows (the notation “%” means “the datatype in question”):

<code>?*? : (% , %) -> %</code>	<code>?/? : (% , %) -> %</code>
<code>?=? : (% , %) -> Boolean</code>	<code>1 : () -> %</code>
<code>?^? : (% , Integer) -> %</code>	<code>coerce : % -> OutputForm</code>
<code>commutator : (% , %) -> %</code>	<code>conjugate : (% , %) -> %</code>
<code>hash : % -> SingleInteger</code>	<code>inv : % -> %</code>
<code>latex : % -> String</code>	<code>one? : % -> Boolean</code>
<code>sample : () -> %</code>	<code>?~=? : (% , %) -> Boolean</code>

Note that (mathematical) equality is specifically defined here. Thus an Axiom **Group** is more than just a mathematical group: it is one in which equality is decidable — certainly not the case in all mathematical groups [52, pp. 430–431]. We discuss this point later.

The axioms are currently mostly defined as a special form of comment in the Axiom source code, and are intended for human use. However Kelsey ([41], see also section 7.1) has shown that it is possible to convert these into Larch [33] traits. Some axioms can also be seen as definitions of the corresponding operations, and are therefore represented in Axiom as **default** definitions, as in the following from **Group**.

```
x:% / y:% == x*inv(y)
conjugate(p,q) == inv(q) * p * q
commutator(p,q) == inv(p) * inv(q) * p * q
one? x == x = 1
_~_(x:%,y:%) : Boolean == not(x=y)
```

(the last line defines the infix operation `~=` meaning “not equal to”). These defaults can be over-ridden by specific types: for example matrix groups might well over-ride the definition of `/`.

Domains are arranged in a multiple-inheritance structure (see [13]), so that in practice **Group** is an extension of **Monoid** by adding the operations `inv`, `/`,

⁴ Because of the inheritance structure of Axiom [13], what we show here is actually built up through several layers of inheritance. Nonetheless, these are the actual operators on an Axiom **Group**, having removed some duplication.

`commutator` and `conjugate` (and the default definitions of the last three), and `Monoid` is an extension of `SemiGroup` by adding `1` and `one?`. Similarly `Rng` (a ring, possibly without unity) is an extension of both `AbelianGroup` and `SemiGroup`, whereas `Ring` is an extension of `Rng` and `Monoid`. Of course, not every object which is both an `AbelianGroup` and a `SemiGroup` is a `Rng`: the distributive axiom has to be satisfied. In the current version of Axiom, this verification is done by the programmer when he declares that a certain type returns a `Rng`, but the work described in section 7.1 shows that this can be automated.

3.1 An Example: Polynomials

By “polynomial” we mean “polynomial in commuting variables”: the task of extending the structures to non-commuting variables is beyond our scope here. The category structure required to support the polynomial world is fairly complex [14], but we will concentrate on only two categories, `PolynomialCategory` (henceforth `POLYCAT`) and its specialisation, `UnivariatePolynomialCategory` (henceforth `UPOLYC`). The former is parameterized by three domains: the coefficient ring, the exponent domain (in particular this will contain the comparison function between exponents, i.e. between monomials, which is vital for Gröbner base [5] applications) and the set of variables (which is usually `Symbol`), whereas `UPOLYC` is parameterized only by the underlying coefficient ring.

`UPOLYC` is a `Ring`, but also supports many polynomial-like operations, such as `degree` and `leadingCoefficient`. The obvious constructors in `UPOLYC` are `dense` and `sparse` univariate polynomials. The latter, being the commonly-used type, are created by the functor `UnivariatePolynomial` (henceforth `UP`), parameterized by the symbol in which they are, and the underlying coefficient ring. If we take the polynomial $x^2y + y + 1$, then in `UP(x,UP(y,INT))` (i.e. $\mathbf{Z}[y][x]$) it would have degree 2 and leading coefficient $y \in \text{UP}(y, \text{INT})$. However, in `UP(y,UP(x,INT))` (i.e. $\mathbf{Z}[x][y]$) it would have degree 1 and leading coefficient $x^2 + 1 \in \text{UP}(x, \text{INT})$. This illustrates the point that, although $\mathbf{Z}[x][y]$ and $\mathbf{Z}[y][x]$ are isomorphic as *abstract* rings, they are not isomorphic as elements of `UPOLYC` — see section 6.1.

There are two main families of functors returning domains in `POLYCAT`. One, typically `Polynomial` (abbreviated `POLY` and parameterized only by the coefficient ring), is *recursive*, i.e. implementing $\mathbf{Z}[x, y]$ in terms of $\mathbf{Z}[y][x]$ and regarding $x^2y + y + 1$ as $x^2y + x^0(y + 1)$ — the sum of two terms, whereas the other is *distributed* [53], regarding the same polynomial as the sum of three terms: x^2y^1 , x^0y^1 and x^0y^0 . It should be emphasised that the internal representation in the recursive case is transparent (apart from issues of efficiency): in both cases the `leadingCoefficient` is 1, and the `leadingMonomial` is x^2y . However, the general distributed form is parameterized not only by the coefficient ring but also the exponent domain (an `OrderedCancellationAbelianMonoid` to be precise), and the ordering function determines the leading monomial. In particular, different ordering functions can make any of the monomials in $x^2yz + xy^2z + xyz^2 + x^3 + y^3 + z^3$ into the leading one. Again, these different domains will be isomorphic as abstract rings, but not as `POLYCATs`.

4 Typing Issues

To take a concrete example, how should a system treat the user input $x+2$, where we assume that x is of type `Symbol` and 2 is of type `PositiveInteger`, in order to produce a sum of type `Polynomial(Integer)`? (Life would of course be simpler in this case if x and 2 were polynomials, but this would create even greater problems when one wanted to specify orderings on `Symbols`, as needed in various Gröbner base applications, or matrices whose size has to be a positive integer.)

The traditional Abstract Data Types view would treat this as an issue of subsorts:

$$\begin{aligned}\text{Symbol} &\subset \text{Polynomial}(\text{Integer}) \\ \text{PositiveInteger} &\subset \text{Integer} \subset \text{Polynomial}(\text{Integer}).\end{aligned}$$

In practice, however, these subsort relationships often have to translate into actual changes of data representation. One obvious reason for this is efficiency, but a more fundamental one is that we do not have, in practice, the neat subsort graph that appears in examples of Abstract Data Type theory. Indeed, the subsort graph in computer algebra is not even finite, so that x has to be viewed as being not only in `Polynomial(Integer)`, but also in `Polynomial(PrimeField(2))`, `Polynomial(PrimeField(3))` etc., as well as

`Polynomial(SquareMatrix(2,Integer))` and many other data types. Given that the lattice of data types is infinite (and can change as the user adds new data types), there is a genuine problem of implementing coercion [24,54]. Having decided that we want to get from type A to type B , how do we get there? Clearly *ad hoc* programming is not the right answer, instead we should appeal to category theory [51] and build a generic mechanism. Suppose we want $2 \in \mathbf{Z}[x][y]$. We could start with $2 \in \mathbf{Z}$, apply $R \rightarrow R[x]$ then $R \rightarrow R[y]$. Equally, we could apply $R \rightarrow R[y]$, then lift each of the coefficients from $\mathbf{Z}$ to $\mathbf{Z}[x]$. This second map might seem perverse, but it has to exist to map $y+1 \in \mathbf{Z}[y]$ to $\mathbf{Z}[x][y]$, and to map $\mathbf{Z}[y]$ to $\text{GF}(29)[y]$. It is clearly desirable that the route taken does not affect the value of the answer (it may affect running time, but that is another question). This issue has been studied within the computer algebra community [59,16,17], but further work on developing a practical consistency check would be highly desirable.

5 Equality

As we have already remarked, an explicit equality operation is a general feature of Axiom's category structure [13], though not all types have to have equality: e.g. types of functions do not. Implicitly, equality is required in many algorithms, e.g. fraction-free [4,15] matrix algorithms. This may also be present in operations such as testing for zero (Axiom's `zero?`) etc. If one considers an Axiom

Field, it is an extension of **GcdDomain**, and therefore has operations such as (normalized⁵)**gcd**. What is **gcd(a,b)** in a field? The correct answer is one, unless $a = b = 0$, when the answer is zero. Hence **zero?** is necessary for implementing **gcd**.

This means that an arbitrary-precision implementation of **R**, e.g. via linear fractional transformations [43] or B -adic arithmetic [6], is not an element of the category **Field** [38], but rather of a weaker structure, viz. a **SemiField** [37], i.e. an Axiom **Field** without equality and the related operations.

There is no requirement in Axiom for data structure equality to be the same as mathematical equality: types for which this is known to be true are said to be **canonical**. This causes substantial problems [41] when interfacing with proof-checkers where the two are assumed to be the same, and a full implementation of, say, Axiom's logic in Larch [33], would need to distinguish between the two kinds of equality.

Another difficulty is presented by non-deterministic (Monte Carlo) equality tests, as arise in many data structures such as straight-line programs [46]. It is common to use a fixed bound for the accuracy of such tests⁶, but in fact the precision required should depend on the depth of nesting: e.g. is 99% accuracy has been requested for the final result, and the algorithm will make 100 tests, these should each be done to 99.99% certainty. This requires significant modifications to the category structure: see [46].

6 Are Abstract Data Types a Complete Model

It is frequently taken as axiomatic that the theory of abstract data types is a (often “the”) correct model for mathematics, in particular for computer algebra. Is this really so obvious?

6.1 The Meanings of “Isomorphic”

We have already seen that $\mathbf{Z}[x][y]$ and $\mathbf{Z}[y][x]$ are isomorphic as abstract **Rings**, but not as polynomial rings (**UPOLYC**), in the sense that the same object in the two rings will have different values for functions like **degree**. The algorithm described in [16] can generate a coercion (a concrete realisation of the ring isomorphism) between them, essentially by deconstructing the data object in the source into its atoms (x , y and integers) and reconstructing the result using the **Ring** operations in the target domain. However, one would clearly not want this operation to be performed automatically. Nor would one want the polynomial $x = y$ to be converted into the string **xy**, since x converts to **x** and on strings, $+$ is concatenation.

⁵ The g.c.d. is only defined up to units — see [14].

⁶ And indeed other probabilistic algorithms such as **prime?**, but that's another story.

6.2 Deconstructors

The example of lists is frequently given in Abstract Data Type theory, with constructors **nil** and **cons** and deconstructors **first** and **rest**, with axioms such as

$$\begin{aligned}\text{first}(\text{cons}(a, b)) &= a \\ \text{rest}(\text{cons}(a, b)) &= b,\end{aligned}$$

and it is frequently said that much the same can be done with fractions, with a constructor **frac** and deconstructors **num** and **den**, and axioms such as

$$\begin{aligned}\text{num}(\text{frac}(a, b)) &= a \\ \text{den}(\text{frac}(a, b)) &= b,\end{aligned}\tag{1}$$

as well as mathematical operations such as

$$\text{frac}(a, b) * \text{frac}(c, d) = \text{frac}(ad + bc, bd).$$

In this view “=” is data structure equality, as we have discussed above, not mathematical equality, and we would need to define the latter by

$$\text{frac}(a, b) \equiv \text{frac}(c, d) = ad = bc.\tag{2}$$

However, this view will be hopelessly inefficient in practice.

Conversely, if one wants mathematical equality, then the equations (1) have to be abandoned as such. One can replace them by equations such as

$$\text{frac}(\text{num}(c), \text{den}(c)) = c,$$

but the simple constructor/deconstructor model is lost.

From the Abstract Data Type point of view, there is essentially only one model of the set of equations using equations (1): the “no cancellation, denominators can be negative” model (call it $\mathbf{Z} \times \mathbf{Z}_{\neq 0}$), which is both initial and final. If we use equation (3) etc., there are a variety of models, ranging from $\mathbf{Z} \times \mathbf{Z}_{\neq 0}$ to $\mathbf{Q}$ depending on the amount of “simplification” that is carried out. If, however, we replace “ $\equiv$ ” in equation (2) by “=”, then there is only one model, $\mathbf{Q}$.

6.3 Operations Depend on Domains

The order-sorted algebra view insists that the signatures should be regular and that an algebra implementing an order-sorted signature should be consistent (meaning that if the same operation is defined with two different signatures, its implementations must agree on the intersection). How reasonable is this in mathematics?

At first sight it seems very reasonable: $2 + 2 = 4$ whether we are thinking of the “2”s as being natural numbers, integers, rational numbers, complex numbers, (trivial) polynomials, or even as matrices. The same is true of most other purely

arithmetic operations. Of course, $2 + 2 = 1$ in $\text{GF}(3)$, but we view $\mathbf{Z}$ and $\text{GF}(3)$ as incomparable sorts, so this is not a problem. When we look at the Axiom polynomial world (see section 3.1), this becomes less true. $\mathbf{Z}[x] \subset \mathbf{Z}[x][y]$, but in the former $x + 1$ has degree one, and in the latter it is really $y^0(x + 1)$ and so has degree zero.

There are some operations, though, for which the domain over which they are computed is fundamental to the result. The classic example [25] is “factor”, since $\text{factor}(x^2 + 1) = x^2 + 1$ in $\mathbf{Z}[x]$, but $(x + i)(x - i)$ in $\mathbf{Z}[i][x]$. Maple gets round this problem by defining **factor** to be the polynomial factorization, and **ifactor** is used if integer factorization is needed, but the cost is, of course, that “factor” is no longer a generic operation. In a statically typed system, such as compiled Axiom/Aldor, there is no real inconsistency in practice, since the **factor** operation called will be from the type of its argument. In an interpreted world, though, it is easy to, and indeed Axiom will, land in an apparent paradox in a user-friendly interpreter in which type coercions (moving to supersorts) occur freely.⁷

```
(1) -> x^2+1
      2
      (1)  x  + 1
                                         Type: Polynomial Integer

(2) -> factor %
      2
      (2)  x  + 1
                                         Type: Factored Polynomial Integer

(3) -> %% 1 + %i - %i
      2
      (3)  x  + 1
                                         Type: Polynomial Complex Integer

(4) -> factor %
      (4)  (x - %i)(x + %i)
                                         Type: Factored Polynomial Complex Integer
```

A similar problem arises in practice with **gcd**, since in a field, the (normalized) g.c.d. of non-zero objects is always one. Hence the following similar example.

```
(6) -> gcd(6,3)
      (6)  3
                                         Type: PositiveInteger

(7) -> gcd(12/2,3)
      (7)  1
                                         Type: Fraction Integer
```

From a theoretical point of view, the difficulty that these operations are not consistent (in the order-sorted algebra sense) can be removed by making **factor** use dependent types, i.e. instead of having signatures like

⁷ In interactive Axiom sessions, % denotes the last result.

```
factor: (f:R) -> Factored(R)
```

the signature could be

```
factor: (R:Ring,f:R) -> Factored(R)
```

however, the author knows of no extension to order-sorted algebras that really allows for this. The user interface problem is hard, and Axiom's solution of (by default) printing the type of the object is probably not the best.

7 Making the Mathematics Explicit

So far we have concentrated on computer algebra systems, and described how they can be structured via Abstract Data Type theory, and some of the little-documented problems of this approach. However, we have not described how the system could check itself, or how truly formal methods could be applied to this area — surely one in which they should be applied. The author is aware of two major approaches using existing computer algebra systems (as opposed to saying “damn the efficiency — full speed ahead with the logic” as an OBJ-based, say, system might).

7.1 The Saint Andrews Approach

This [19,20] consists of using an independent theorem-prover (currently Larch [33]) to prove the “proof obligations” inherent in Axiom. Firstly, it is necessary to make the axioms of Axiom's categories explicit, which was done in [41]. There are then two kinds of proof obligations.

- The statement that a domain really does belong to a certain category has to be checked. This means that the operators of the domain (defined by certain pieces of program, in that domain or in the domain it inherits from, and recursively) must satisfy the axioms of the category. Significant progress in this direction was demonstrated in [18], working from an intermediate (typed) form from the Aldor compiler, rather than from the source, to ensure consistency of typing algorithm. Of course, if the domain is parameterized, one will need the axioms satisfied by the categories of the parameter domains.
- The statements that certain properties of domains in a category (often needed in the proofs of the first kind) do follow from the axioms need to be checked. Significant progress in this direction was demonstrated in [41]. However, the fact that Larch believes in data-structure equality rather than mathematical equality proved to be somewhat of an obstacle, and further work is needed, especially as the documentation of Axiom is not always clear on this point: put another way, it is not always clear to the reader when $a = b \implies f(a) = f(b)$ (see section 5).

In practice, of course, the two interact, and construction of a fully-fledged interface with a theorem-prover would be a large task.

7.2 The Kent Approach

This approach [48,57] is based on the “propositions as types” school [47], and uses the Aldor⁸ type checker as the proof checker. Having defined (a non-trivial task) a type I with signature

$I(A:\text{BasicType}, a:A, b:A) : \text{Type}$

then a **Monoid** M must, as well as implementing the usual operations ($=$, $*$ and 1), also implement operations such as

```
leftUnit  : (m:M)          -> I(M,m,1*m)
rightUnit : (m:M)          -> I(M,1*m,m)
assoc     : (m:M,n:M,p:M) -> I(M,m*(n*p),(m*n)*p)
```

These operations are not *used* as such, it is the fact that they *compile* (i.e. type-check) that is the proof that the domain so defined actually is a **Monoid** in terms of satisfying the usual monoid axioms as well as exporting the usual monoid operations. It is worth noting that this approach depends heavily on the Aldor language’s support for *dependent types* [50], e.g. the type of the second and third arguments to I is given as the first argument.

8 Conclusion

The first thing to be said is that basing a computer algebra system on Abstract Data Type theory does seem to work. This is shown by the take-up of Axiom among pure mathematicians who *do* wish to add new data types to the system, and by the progress made in verifying Axiom (see the previous section).

However, Axiom and Magma cannot be said to be “pure” Abstract Data Type systems. Building them has thrown up many research issues, of which we have highlighted the following.

1. The problem of coercion (see section 4).
2. (closely related to the above) the meaning of “isomorphic” (see section 6.1).
3. The interplay between mathematical equality and data-structure equality (see section 5).
4. The proper handling of operators such as **factor**, whose action depends on the *type* of the arguments, as well as the value, and therefore is contrary to order-sorted algebra (see section 6.3).

References

1. Abbott,J.A., Bradford,R.J. & Davenport,J.H., The Bath Algebraic Number Package. Proc. SYMSAC 86, ACM, New York, 1986, pp. 250–253.
2. Abdali,S.K., Cherry,G.W. & Soiffer,N., An Object-oriented Approach to Algebra System Design. Proc. SYMSAC 86, ACM, New York, 1986, pp. 24–30.

⁸ Aldor [58] is the new programming language for Axiom.

3. Abdali, S.K., Cherry, G.W. & Soiffer, N., A Smalltalk System for Algebraic Manipulation. *Proc. OOPSLA 86 (SIGPLAN Notices* **21** (1986) 11) pp. 277–283.
4. Bareiss, E.H., Sylvester's Identity and Multistep Integer-preserving Gaussian Elimination. *Math. Comp.* **22** (1968) pp. 565–578. Zbl. 187,97.
5. Becker, T. & Weispfenning, V. (with H. Kredel), Groebner Bases. A Computational Approach to Commutative Algebra. Springer Verlag, Graduate Texts in Mathematics **141**, 1993.
6. Boehm, H.-J., Cartwright, R., Riggle, M. & O'Donnell, M.J., Exact Real Arithmetic: A Case Study in Higher Order Programming. *Proc. LISP & Functional Programming (ACM, 1986)* pp. 162–173.
7. Bogen, R.A. et al., MACSYMA Reference Manual (version 9). M.I.T. Laboratory for Computer Science, Cambridge, Mass., 1977.
8. Bosma, W., Cannon, J. & Matthews, G., Programming with algebraic structures: design of the Magma language. *Proc. ISSAC 1994, ACM, New York, 1994*, pp. 52–57.
9. Bosma, W., Cannon, J. & Playoust, C., The Magma algebra system. I: The user language. *J. Symbolic Comp.* **24** (1997) pp. 235–265. Zbl. 898.68039.
10. Bradford, R.J., Hearn, A.C., Padget, J.A. & Schrufer, E., Enlarging the REDUCE Domain of Computation. *Proc. SYMSAC 86, ACM, New York, 1986*, pp. 100–106.
11. Comon, H., Lugiez, D. & Schnoebelen, P., A rewrite-based type discipline for a subset of computer algebra. *J. Symbolic Comp.* **11** (1991) pp. 349–368.
12. Coppo, M., An extended polymorphic type system for applicative languages. *Proc. MFCS 80 (Springer Lecture Notes in Computer Science, Springer-Verlag, Berlin-Heidelberg-New York)*, pp. 194–204.
13. Davenport, J.H. & Trager, B.M., Scratchpad's View of Algebra I: Basic Commutative Algebra. *Proc. DISCO '90 (Springer Lecture Notes in Computer Science Vol. 429, ed. A. Miola)*, Springer-Verlag, 1990, pp. 40–54. A revised version is in Axiom Technical Report ATR/1, Nag Ltd., December 1992.
14. Davenport, J.H., Gianni, P. & Trager, B.M., Scratchpad's View of Algebra II: A Categorical View of Factorization. *Proc. ISSAC 1991 (ed. S.M. Watt)*, ACM, New York, pp. 32–38. A revised version is in Axiom Technical Report ATR/2, Nag Ltd., December 1992.
15. Dodgson, C.L., Condensation of determinants, being a new and brief method for computing their algebraic value. *Proc. Roy. Soc. Ser. A* **15** (1866) pp. 150–155.
16. Doye, N.J., Order Sorted Computer Algebra and Coercions. Ph.D. Thesis, University of Bath, 1997. <http://www.bath.ac.uk/~ccsnjd/research/phd.ps>
<http://www.nic.uklinux.net/research/phd.ps>
17. Doye, N.J., Automated Coercion for Axiom. *Proc. ISSAC 1999 (ed. S. Dooley)*, ACM, New York, 1999, pp. 229–235.
18. Dunstan, M.N., Larch/Aldor - A Larch BSL for AXIOM and Aldor. Ph.D. Thesis, University of St. Andrews, 1999.
19. Dunstan, M., Kelsey, T., Linton, S. & Martin, U., Lightweight Formal Methods for Computer Algebra Methods. *Proc. ISSAC 1998 (ed. O. Gloor)*, ACM, New York, 1998, pp. 80–87.
20. Dunstan, M., Kelsey, T., Linton, S. & Martin, U., Formal methods for extensions to CAS. *FM '99 Vol. II (Springer Lecture Notes in Computer Science Vol. 1709, ed. J.W.J. Wing, J. Woodcock & J. Davies)*, Springer-Verlag, 1999, pp. 1758–1777.
21. Ehrig, H.-D., On the Theory of Specification, Implementation and Parameterization of Abstract Data Types. *J. ACM* **29** (1982) pp. 206–227. MR 83g:68030.

22. Ehrig, H. & Mahr, B., Fundamentals of Algebraic Specification 1: Equations and Initial Semantics. EATCS Monographs in Theoretical Computer Science 6, Springer-Verlag, Berlin, 1985.
23. Faugère, J.-C., Gianni, P., Lazard, D. & Mora, T., Efficient Computation of Zero-Dimensional Gröbner Bases by Change of Ordering. *J. Symbolic Comp.* **16** (1993) pp. 329–344.
24. Fortenbacher, A., Efficient Type Inference and Coercion in Computer Algebra. Proc. DISCO '90 (Springer Lecture Notes in Computer Science Vol. 429, ed. A. Miola) pp. 56–60.
25. Fröhlich, A. & Shepherdson, J.C., Effective Procedures in Field Theory. *Phil. Trans. Roy. Soc. Ser. A* **248** (1955–6) pp. 407–432. Zbl. 70,35.
26. Goguen, J.A. & Malcolm, G. (eds.), Software Engineering with OBJ: algebraic specification in action. Kluwer, 2000.
27. Goguen, J.A. & Meseguer, J., Order-sorted Algebra I: Equational deduction for multiple inheritance, polymorphism and partial operations. *Theor. Comp. Sci.* **105** (1992) pp. 217–293.
28. Goguen, J.A., Thatcher, J.W., Wagner, E.G. & Wright, J.B., A Junction Between Computer Science and Category Theory I: Basic Concepts and Examples (Part 1). IBM Research RC 4526, 11 September 1973.
29. Goguen, J.A., Thatcher, J.W., Wagner, E.G. & Wright, J.B., An Introduction to Categories, Algebraic Theories and Algebras. IBM Research RC 5369, 16 April 1975.
30. Goguen, J.A., Thatcher, J.W., Wagner, E.G. & Wright, J.B., A Junction Between Computer Science and Category Theory I: Basic Concepts and Examples (Part 2). IBM Research RC 5908, 18 March 1976.
31. Goguen, J.A., Thatcher, J.W., Wagner, E.G. & Wright, J.B., Initial Algebra Semantics and Continuous Algebras. *J. ACM* **24** (1977) pp. 68–95.
32. Gruntz, D. & Monagan, M., Introduction to Gauss. *MapleTech: The Maple Technical Newsletter*, Issue 9, Spring 1993, pp. 23–49.
33. Guttag, J.V. & Horning, J.J., Larch: Languages and Tools for Formal Specification. Texts and Monographs in Computer Science, Springer-Verlag, 1993.
34. Hearn, A.C., REDUCE User's Manual, Version 3.4, July 1991. RAND Corporation Publication CP-78.
35. Hearn, A.C. & Schrüfer, E., An Order-Sorted Approach to Algebraic Computation. Proc. DISCO '93 (ed. A. Miola), Springer Lecture Notes in Computer Science 722, Springer-Verlag, 1993, pp. 134–144.
36. Hohfeld, B., Correctness Proofs of the Implementation of Abstract Data Types. Proc. EUROCAL 85, Vol. 2 (Springer Lecture Notes in Computer Science Vol. 204, Springer-Verlag, 1985) pp. 446–447.
37. Hur, N., A Symbolic and Numeric Approach to Real Number Computation. Draft Ph.D. Thesis, University of Bath, 2000.
38. Hur, N. & Davenport, J.H., An Exact Real Algebraic Arithmetic with Equality Determination. Proc. ISSAC 2000 (ed. C. Traverso), pp. 169–174.
39. Jenks, R.D. & Sutor, R.S., AXIOM: The Scientific Computation System. Springer-Verlag, New York, 1992.
40. Jenks, R.D. & Trager, B.M., A Language for Computational Algebra. Proc. SYMSAC 81, ACM, New York, 1981, pp. 6–13. Reprinted in *SIGPLAN Notices* **16** (1981) No. 11, pp. 22–29.
41. Kelsey, T.W., Formal Methods and Computer Algebra: A Larch Specification of Axiom Categories and Functors. Ph.D. Thesis, St. Andrews, 2000.

42. Kounalis,E., Completeness in Data Type Specifications. Proc. EUROCAL 85, Vol. 2 (Springer Lecture Notes in Computer Science Vol. 204, Springer-Verlag, 1985) pp. 348–362.
43. Ménéssier-Morain,V., Arithmétique exacte, conception, algorithmique et performances d’une implémentation informatique en précision arbitraire. Thèse, Université Paris 7, Dec. 1994.
44. Monagan,B., Gauss: A Parameterized Domain of Computation System with Support for Signature Functions. Proc. DISCO ’93 (ed. A. Miola), Springer Lecture Notes in Computer Science 722, Springer-Verlag, 1993, pp. 81–94.
45. Musser,D.R. & Kapur,D., Rewrite Rule Theory and Abstract Data Type Analysis. Proc. EUROCAM 82 (Springer Lecture Notes in Computer Science 144, Springer-Verlag, Berlin-Heidelberg-New York, 1982), pp. 77–90. MR 83m:68022.
46. Naylor,W.A., Polynomial GCD Using Straight Line Program Representation. Ph.D. Thesis, University of Bath, 2000.
47. Nordström,B., Petersson,K. & Smith,J.M., Programming in Martin-Löf’s Type Theory — An Introduction. OUP, 1990.
48. Poll,E. & Thompson,S., Integrating Computer Algebra and Reasoning through the Type System of Aldor. Proc. Frontiers of Combining Systems: FroCoS 2000 (Springer Lecture Notes in Computer Science 1794, Springer-Verlag, 2000, ed. H. Kirchner & C. Ringeissen).
49. Rector,D.L., Semantics in Algebraic Computation. Computers and Mathematics (ed. E. Kaltofen & S.M. Watt), Springer-Verlag, 1989, pp. 299–307.
50. Reynaud,J.-C., Putting Algebraic Components Together: A Dependent Type Approach. Proc. DISCO ’90 (Springer Lecture Notes in Computer Science Vol. 429, ed. A. Miola) pp. 141–150.
51. Reynolds,J.C., Using Category Theory to Design Implicit Conversions and Generic Operators. Semantics-Directed Compiler Generation (Springer Lecture Notes in Computer Science 94, ed. N.D. Jones), 1980, pp. 211–258.
52. Rotman,J.J., An Introduction to the Theory of Groups. Springer Graduate Texts in Mathematics 148, Springer-Verlag, 1995.
53. Stoutemyer,D.R., Which Polynomial Representation is Best: Surprises Abound. Proc. 1984 MACSYMA Users’ Conference (ed. V.E. Golden), G.E., Schenectady, pp. 221–243.
54. Sutor,R.S. & Jenks,R.D., The Type Inference and Coercion Facilities in the Scratchpad II Interpreter. Proc. SIGPLAN ’87 Symp. Interpreters and Interpretive Techniques (*SIGPLAN Notices* **22** (1987) 7) pp. 56–63.
55. Thatcher,J.W., Wagner,E.G. & Wright,J.B., Notes on Algebraic Fundamentals for Theoretical Computer Science. Foundations of Computer Science III (ed. J.W. de Bakker & J. van Leeuwen), Mathematical Centre Tract 109, Amsterdam, 1979.
56. Thatcher,J.W., Wagner,E.G. & Wright,J.B., Data Type Specification: Parameterization and the Power of Specification Techniques. *ACM TOPLAS* **4** (1982) pp. 711–732.
57. Thompson,S., Logic and dependent types in the Aldor Computer Algebra System. To appear in Proc. Calculemus 2000.
58. Watt,S.M., Broadbery,P.A., Dooley,S.S., Iglio,P., Morrison,S.C., Steinbach,J.M. & Sutor,R.S., Axiom Library Compiler User Guide. NAG Ltd., Oxford, 1994.
59. Weber,A., Algorithms for type inference with coercions. Proc. ISSAC 1994, ACM, New York, 1994, pp. 324–329.
60. Zippel,R.E., The Weyl Computer Algebra Substrate. Proc. DISCO ’93 (ed. A. Miola), Springer Lecture Notes in Computer Science 722, Springer-Verlag, 1993, pp. 303–318.

What Do We Learn from Experimental Algorithmics?*

Camil Demetrescu¹ and Giuseppe F. Italiano²

¹ Dipartimento di Informatica e Sistemistica,
Università di Roma “La Sapienza”,
Via Salaria 113, 00198 Roma, Italy. Email: demetres@dis.uniroma1.it,
URL: <http://www.dis.uniroma1.it/~demetres/>

² Dipartimento di Informatica, Sistemi e Produzione,
Università di Roma “Tor Vergata”,
Via di Tor Vergata 110, 00133 Roma, Italy. Email: italiano@disp.uniroma2.it,
URL: <http://www.info.uniroma2.it/~italiano/>

Abstract. Experimental Algorithmics is concerned with the design, implementation, tuning, debugging and performance analysis of computer programs for solving algorithmic problems. It provides methodologies and tools for designing, developing and experimentally analyzing efficient algorithmic codes and aims at integrating and reinforcing traditional theoretical approaches for the design and analysis of algorithms and data structures.

In this paper we survey some relevant contributions to the field of Experimental Algorithmics and we discuss significant examples where the experimental approach helped in developing new ideas, in assessing heuristics and techniques, and in gaining a deeper insight about existing algorithms.

1 Introduction

Experiments play a crucial role in many scientific disciplines. In Natural Sciences, for instance, researchers have been extensively running experiments to learn certain aspects of nature and to discover unpredictable features of its internal organization. In Theoretical Computer Science we use mathematical tools for analyzing and predicting the behavior of algorithms. For over thirty years, asymptotic worst-case analysis has been the main model in the design of efficient algorithms, proving itself to yield enormous advantages in comparing and characterizing their behavior, and leading to major algorithmic advances.

In recent years, many areas of theoretical computer science have shown growing interest in solving problems arising in real world applications, experiencing

* This work has been partially supported by the European Commission (Project ALCOM-FT), by the Italian Ministry of University and Scientific Research (Project “Algorithms for Large Data Sets: Science and Engineering”) and by CNR, the Italian National Research Council.

a remarkable shift to more application-motivated research. The new demand for algorithms that are of practical utility has led researchers to take advantage from the experimental method as a tool both for refining and reinforcing the theoretical analysis of algorithms, and for developing and assessing heuristics and programming techniques for producing codes that are efficient in practice. This is clear from the surge of investigations that compare and analyze experimentally the behavior of algorithms, and from the rise of a new research area called *Experimental Algorithmics*.

Experimental Algorithmics is concerned with the design, implementation, tuning, debugging and performance analysis of computer programs for solving algorithmic problems and aims at joining the experimental method with more traditional theoretical approaches for the design and analysis of algorithms and data structures.

1.1 Goals

Major goals of Experimental Algorithmics are:

- Defining standard methodologies for algorithm engineering.
- Devising software systems to support the process of implementation, debugging and empirical evaluation of algorithms. In particular, programming environments, high-level debuggers, visualization and animation tools, and testing and simulation environments.
- Identifying and collecting problem instances from the real world and providing generators of synthetic test sets for experimental evaluation of algorithms.
- Providing standard and well documented libraries that feature efficient implementations of algorithms and data structures.
- Performing empirical studies for comparing actual relative performance of algorithms so as to identify the best ones for use in a given application. This may lead to the discovery of algorithm separators, i.e., families of problem instances for which the performances of solving algorithms are clearly different. Other important results of empirical investigations include assessing heuristics for hard problems, characterizing the asymptotic behavior of complex algorithms, discovering the speed-up achieved by parallel algorithms and studying memory hierarchy and communication effects on real machines, helping in performance prediction and finding bottlenecks in real applications, etc.
- Last, but not least, encouraging fruitful cooperation between theoreticians and practitioners.

1.2 Motivations

The main motivations for recurring to experiments for analyzing and drawing on-the-road conclusions about algorithms have already been pointed out by different authors [3,16,19,26]. Among many, we cite the following:

- Many authors call “Asymptopia” the range of problem instances for which an algorithm exhibits clear asymptotic behavior. Unfortunately, for certain algorithms, “Asymptopia” may include only huge problem instances, far beyond the needs of any reasonable practical application. This means that, due to the high constants hidden in the analysis, theoretical bounds may fail to describe the behavior of algorithms on many instances of practical interest. As a typical example, the experimental study in [27] shows that the minimum spanning tree algorithm of Fredman and Tarjan improves in practice upon the classical Prim’s algorithm only for huge dense graphs with more than one million nodes.
- The situation may be even worse: constants hidden in the asymptotic time bounds may be so large as to prevent any practical implementation from running to completion. The Robertson and Seymour cubic-time algorithm for testing if a graph is a minor of another graph provides an extreme example: as a matter of fact, no practical implementation can face the daunting 10^{150} constant factor embedded in the algorithm, and no substantial improvement has been proposed yet.
- Many algorithms typically behave much better than in the worst case, so considering just the worst-case bounds may lead to underestimate their practical utility. A typical example is provided by the simplex algorithm for solving linear programs, whose asymptotic worst-case time bound is exponential, while its running time seems to be bounded by a low-degree polynomial in many real-world instances.
- Many practical applications require solving NP-hard problems, for which asymptotic analyses do not provide satisfactory answers about the utility of a solving algorithm.
- There are algorithms for which no tight asymptotic bounds on the running time or on the quality of returned solutions have been theoretically proved. For instance, the experimental study proposed in [14] compares the quality of the solutions returned by different approximation algorithms for the problem of minimizing edge crossings in the drawings of bipartite graphs. It reports that an algorithm with no theoretically proved constant approximation ratio returns in practice solutions with less crossings than algorithms with small, constant approximation ratio.
- New algorithmic results often rely on previous ones, and devising them only at theoretical level may lead to a major problem: researchers who would eventually come up with a practical implementation of their results may be required to code several layers of earlier unimplemented complex algorithms and data structures, and this task may be extremely difficult.
- Adding ad-hoc heuristics and local hacks to the code may dramatically improve the practical performances of some algorithms, although they do not affect the theoretical asymptotic behavior. Many clear examples are addressed in the literature: in Section 3.1 we will discuss in detail the implementation issues of the Preflow-push Maximum Flow algorithm of Goldberg and Tarjan.

- Performing experiments on a good collection of test problems may help in establishing the correctness of a code. In particular, collecting problem instances on which a code has exhibited buggy behavior may be useful for testing further implementations for the same problem.
- Practical indicators, such as implementation constant factors, real-life bottlenecks, locality of references, cache effects and communication complexity may be extremely difficult to predict theoretically, but can be measured by means of experiments.

1.3 Common Pitfalls

Unfortunately, as in any empirical science, it may be sometimes difficult to draw general conclusions about algorithms from experiments. Common pitfalls, often experienced by researchers in their studies, seem to be:

- Dependence of empirical results upon the experimental setup:
 - Architecture of the running machine: memory hierarchy, CPU instructions pipelining, CISC vs RISC architectures, CPU and data bus speed are technical issues that may substantially affect the execution performance.
 - Operating system: CPU scheduling, communication management, I/O buffering and memory management are also important factors.
 - Encoding language: features such as built-in types, data and control flow syntactic structures and language paradigm should be taken into account when choosing the encoding language. Among others, C++, C and Fortran are most commonly used in this context. However, we point out that powerful C++ features such as method invocations, overloading of functions and operators, overriding of virtual functions, dynamic casting and templates may introduce high hidden computation costs in the generated machine code even using professional compilers.
 - Compiler's optimization level: memory alignment, register allocation, instruction scheduling, repeated common subexpression elimination are the most common optimization issues.
 - Measuring of performance indicators: time measuring may be a critical point in many situations including profiling of fast routines. Important factors are the granularity of the time measuring function (typically 1 μsec to 10 $msec$, depending upon the platform), and whether we are measuring the real elapsed time, the time used by the user's process, or the time spent by the operating system to do I/O, communication or memory management.
 - Programming skills of implementers: the same algorithm implemented by different programmers may lead to different conclusions on its practical performances. Moreover, even different successive refined implementations coded by the same programmer may greatly differ from each other.
 - Problem instances used in the experiments: the range of parameters defining the test sets used in the experiments and the structure of the

problem instances themselves may lead to formulate specific conclusions on the performance of algorithms without ensuring generality. Another typical pitfall in this context consists of testing codes on data sets representing classes that are not broad enough. This may lead to inaccurate performance prediction. An extreme example is given by the Netgen problem instances for the minimum cost flow problem [20] that were used to select the best code for a multicommodity flow application [23]. That code was later proved to behave much slower than several other codes on real-life instances by the same authors of [23]. In general, it has been observed that some algorithms behave quite differently if applied on real-life instances and on randomly generated test sets. Linear programming provides a well known example.

- Difficulty in separating the behavior of algorithms: it is sometimes hard to identify problem instances on which the performance of two codes is clearly distinguishable. In general, good algorithm separators are problem families on which differences grow with the problem size [16].
- Unreproducibility of experimental results: possibly wrong, inaccurate or misleading conclusions presented in experimental studies may be extremely difficult to detect if the results are not exactly and independently reproducible by other researchers.
- Modularity and reusability of the code: modularity and reusability of the code seem to conflict with any size and speed optimization issues. Usually, special implementations are difficult to reuse and to modify because of hidden or implicit interconnections between different parts of the code, often due to sophisticated programming techniques, tricks and hacks which they are based on, but yield to the best performances in practice. In general, using the C++ language seems to be a good choice if the goal is to come up with a modular and reusable code because it allows defining clean, elegant interfaces towards (and between) algorithms and data structures, while C is especially well suited for fast, compact and highly optimized code.
- Limits of the implementations: many implementations have strict requirements on the size of the data they deal with, e.g., work only with small numbers or problem instances up to a certain maximum size. It is important to notice that ignoring size limits may lead to substantially wrong empirical conclusions, especially in the case where the used implementations, for performance reasons, do not explicitly perform accurate data size checking.
- Numerical robustness: implementations of computational geometry algorithms may typically suffer from numerical errors due to the finite-precision arithmetic of real computing machines.

Although it seems there is no sound and generally accepted solution to these issues, some researchers have proposed accurate and comprehensive guidelines on different aspects of the empirical evaluation of algorithms maturated from their own experience of in the field (see, for example [3,16,19,26]). The interested reader may find in [24] an annotated bibliography of experimental algorithmics sources addressing methodology, tools and techniques.

In the remainder of this paper, we first survey some relevant contributions to the area of Experimental Algorithmics, pointing out their impact in the field. Then we discuss two case studies and the lessons that we can learn from them.

2 Tools for Experimental Analysis of Algorithms

The increasing attention of the algorithmic community to the experimental evaluation of algorithms is producing, as a side effect, several tools whose target is to offer a general-purpose workbench for the experimental validation and fine-tuning of algorithms and data structures. In particular, software libraries of efficient algorithms and data structures, collections and generators of test sets for experimental evaluation, and software systems for supporting the implementation process are relevant examples of such an effort.

2.1 Software Libraries

The need for robust and efficient implementations of algorithms and data structure is one of the main motivations for any experimental work in the field of Algorithm Engineering. Devising fast, well documented, reliable, and tested algorithmic codes is a key aspect in the transfer of theoretical results into the setting of applications, but it is fraught with many of the pitfalls described in Section 1. Without claim of completeness, we survey some examples of such an effort.

LEDA. LEDA (Library of Efficient Data Types and Algorithms) is a project that aims at building a library of efficient data structures and algorithms used in combinatorial computing [25]. It provides a sizable collection of data types and algorithms in a form which allows them to be used by non-experts. The authors started the LEDA project in 1989 as an attempt to bridge the gap between algorithm research, teaching and implementation. The library is written in C++ and features efficient implementations of most of the algorithms and data types described in classical text books such as [1,2] and [11]. Besides, it includes classes for building graphic user interfaces and for I/O, error handling and memory management. In particular, LEDA contains:

- Data structures for arrays, stacks, queues, lists, sets, dictionaries, priority queues, ordered sequences, partitions, lines, points, planes, dynamic trees and directed, undirected and planar graphs. Efficient implementations are given for each of the data types, e.g., Fibonacci Heaps and Redistributive Heaps for priority queues, Red-black Trees and Dynamic Perfect Hashing for dictionaries, etc. Moreover, the library features generators for several different classes of graphs including complete, random, bipartite, planar, grid and triangulated graphs, and testers for many graph properties, including planarity testing.

- Basic algorithms on graphs and networks and computational geometry algorithms such as topological sort, graph visits, connected, strongly connected and biconnected components, as well as transitive closure, shortest paths, maxflow, min-cost flow, min cut, maximum cardinality matching, minimum spanning tree, st-numbering, and others.
- C++ classes for dealing with windows, widgets and menus, drawing and event handling routines, and other miscellaneous utility classes.

Additional LEDA Extension Packages (LEP) have been developed by different researchers, and include Dynamic Graph Algorithms, Abstract Voronoi Diagrams, D-Dimensional Geometry, Graph Iterators, Parametric Search, SD-Trees and PQ-Trees.

LEDA can be used with almost any C++ compiler (g++, CC, xlc, cxx, Borland, MSVC++, Watcom). It is currently being developed for commercial purposes, but can be used freely for academic research and teaching. LEDA is available over the Internet at the URL: <http://www.mpi-sb.mpg.de/LEDA/>.

The design of the LEDA library is heavily based upon features of the C++ language and the library itself is intended to be a flexible and general purpose tool: for this reason, programs based on it tend to be less efficient than special implementations. However, LEDA is often used as a practical framework for empirical studies in the field of Experimental Algorithmics.

Stony Brook Algorithm Repository. The Stony Brook Algorithm Repository is a comprehensive collection of algorithm implementations for over seventy of the most fundamental problems in combinatorial algorithms. The repository is accessible via WWW at the URL: <http://www.cs.sunysb.edu/algorithm/>. Problems are classified according to the following categories:

- Data Structures
- Numerical Problems
- Combinatorial Problems
- Graph Problems – polynomial-time problems
- Graph Problems – hard problems
- Computational Geometry
- Set and String Problems

The repository features implementations coded in in different programming languages, including C, C++, Fortran, Lisp, Mathematica and Pascal. Also available are some input data files including airplanes routes and schedules, a list of over 3000 names from several nations, and a subgraph of the Erdos-number author connectivity graph.

According to a study on WWW hits to the Stony Brook Algorithm Repository site recorded over a period of ten weeks [29], most popular problems were shortest paths, traveling-salesman, minimum spanning trees as well as triangulations and graph data structures. On the opposite, least popular problems, among others, were determinants, satisfiability and planar drawing.

2.2 Test Sets

Collecting, designing and generating good problem instances for algorithm evaluation is a fundamental task in Experimental Algorithmics. For this reason, much effort has been put in collecting and defining standard test sets and generators for both specific problems and general purpose applications. The Stanford GraphBase and a recent CATS project are two examples of such an effort.

Stanford GraphBase. The Stanford GraphBase [21] is a collection of datasets and computer programs that generate and examine a wide variety of graphs and networks. Differently from other collections of test sets, the Stanford GraphBase consists of small building blocks of code and data and is less than 1.2 megabytes altogether. Data files include the following:

- **econ.dat**: numerical data representing the input/output structure of the entire United States economy in 1985.
- **games.dat**: information about prominent football teams of U.S. colleges or universities that played each other during 1990.
- **miles.dat**: contains highway distances between 128 North American cities.
- **lisa.dat**: contains a digitized version of Leonardo da Vinci’s famous painting, *Mona Lisa*.
- **anna.dat**, **david.dat**, **homer.dat**, **huck.dat** and **jean.dat**: contain “digested” versions of five classic works of literature, namely Tolstoj’s *Anna Karenina*, Dickens’s *David Copperfield*, Homer’s *Iliad*, Twain’s *Huckleberry Finn*, and Hugo’s *Les Misérable*.
- **words.dat**: contains a dictionary of 5757 words, representing every five-letter word of English, compiled by Donald E. Knuth over a period of 20 years.

Several instance generators included in the package are designed to convert these data files into a large variety of interesting test sets that can be used to explore combinatorial algorithms. Other generators produce graphs with a regular structure or random instances.

CATS. CATS (Combinatorial Algorithms Test Sets) [17] is a project whose mission is to facilitate experimental research by standardizing common benchmarks, providing a mechanism for their evolution, and making them easily accessible and usable. The project aims at identifying significant open questions in the design of good test sets and the assessment of performance of existing algorithms. Other goals are to facilitate algorithm selection for applications by characterizing subproblems and the behavior of competitive algorithms on these subproblems, and to encourage the development of high-quality implementations of advanced algorithms and data structures. CATS currently features an archive of application data, synthetic data and generators of instances for problems such as Maximum Flow and Minimum Spanning Tree. More information about CATS is available at the URL: <http://www.jea.acm.org/CATS/>.

2.3 Software Systems

Libraries are just collections of subroutines that usually provide no interactive environment for developing and experimenting with algorithms. Indeed, the need for software systems such as editors for test sets and development, debugging, visualization and analysis tools has grown continuously in the last two decades. As a matter of fact, such systems proved themselves to yield consistent and valuable support in all phases of the algorithm implementation process. In the following, we briefly address some examples of research effort in this context.

Algorithm Animation Systems. Many software systems in the algorithmic area have been designed with the goal of providing specialized environments for algorithm animation. According to a standard definition [32], algorithm animation is a form of high-level software visualization that uses interactive graphics to enhance the development, presentation, and understanding of computer programs. Systems for algorithm animation have matured significantly since the rise of modern computer graphic interfaces due to their relevance in many applications. Thanks to the capability of conveying a large amount of information in a compact form that is easily perceivable by a human observer, algorithm animation is a powerful tool for understanding the behavior of algorithms and testing their correctness on specific test sets. Actually, visual debugging techniques often help discover both errors due to a wrong implementation of an algorithm and, at a higher level of abstraction, errors due to an incorrect design of the algorithm itself. Sometimes, algorithm animation can help in designing heuristics and local improvements in the code difficult to figure out theoretically. In Section 3.1 we will show an animation example that yields sharp clues to the utility of heuristics for improving the practical performances of an algorithm for solving the maximum flow problem.

Dozens of algorithm animation systems have been developed in the last two decades. The area was pioneered in the 80's by the systems Balsa [6] and Zeus [7]. Concerning other tools, we mention Tango [30], Polka [31], UWPI [18], ZStep95 [32], TPM [15], Pavane [28], Leonardo [12], Eliot [22] and Catai [8].

Computing Environments. Among others, we cite LINK, a software system developed at the Center for Discrete Mathematics and Theoretical Computer Science (DIMACS). LINK has been designed to be a general-purpose, extendible computing environment in which discrete mathematical objects representing real world problems can be manipulated and visualized. The system features a full Scheme interpreter with access to the Tk graphics toolkit and a graphic user interface for creation, manipulation, loading, and storing of graphs, hypergraphs, and their attributes. However, due to the interpretive approach to graphics, the system is not suited for visualizing large data sets.

The LINK project has been started to encourage experimentation with algorithms and properties of graphs, and has been designed primarily as an educational and research tool. However, its development has been discontinued in the last years. The interested reader may find more information about LINK at the URL: <http://dimacs.rutgers.edu/~berryj/LINK.html>.

3 Case Studies

3.1 Maximum Flow

The maximum flow problem, first introduced by Berge and Ghouila-Houri in [4], is a fundamental problem in combinatorial optimization that arises in many practical applications. Examples of the maximum flow problem include determining the maximum steady-state flow of petroleum products in a pipeline network, cars in a road network, messages in a telecommunication network, and electricity in an electrical network. Given a capacited network $G = (V, E, c)$ where V is the set of nodes, E is the set of edges and c_{xy} is the *capacity* of edge $(x, y) \in E$, the maximum flow problem consists of computing the maximum amount of flow that can be sent from a given source node s to a given sink node t without exceeding the edge capacities. A flow assignment is a function f on edges such that $f_{xy} \leq c_{xy}$, i.e., edge capacities are not exceeded, and for each node v (bar the source s and the sink t), $\sum_{(u,v) \in E} f_{uv} = \sum_{(v,w) \in E} f_{vw}$, i.e., the assigned incoming flows and the outgoing flows are equal. Usually, it is required to compute not only the maximum amount of flow that can be sent from the source to the sink in a given network, but also a flow assignment that achieves that amount.

Several methods for computing a maximum flow have been proposed in the literature. In particular, we mention the network simplex method proposed by Dantzig [13], the augmenting path method of Ford and Fulkerson, the blocking flow of Diniz, and the push-relabel technique of Goldberg and Tarjan [2].

The push-relabel method, that made it possible to design the fastest algorithms for the maximum flow problem, sends flows locally on individual edges (push operation), possibly creating flow excesses at nodes, i.e., a *preflow*. A preflow is just a relaxed flow assignment such that for some nodes, called *active nodes*, the incoming flows may overcome the outgoing flows. The push-relabel algorithms work by progressively transforming the preflow into a maximum flow by dissipating excesses of flow held by active nodes that either reach the sink or return back to the source. This is done by repeatedly selecting a current active node according to some selection strategy, pushing as much exceeding flows as possible towards adjacent nodes that have a lower estimated distance from the sink paying attention not to exceed the edge capacities, and then, if the current node is still active, updating its estimated distance from the sink (relabel operation). Whenever an active node cannot reach the sink anymore as no path to the sink remains with some residual unused capacity, its distance progressively increases due to relabel operations until it gets greater than n : when this happens, it starts sending flows back towards the source, whose estimated distance is initially forced to n . This elegant solution makes it possible to deal with both sending flows to the sink and draining undeliverable excesses back to the source through exactly the same push/relabel operations. However, as we will see later, if taken “as is” this solution is not so good in practice.

Two aspects of the push-relabel technique seem to be relevant with respect to the running time: (1) the selection strategy of the current active node, and (2) the way estimated distances from the sink are updated by the algorithm.

The selection strategy of the current active node has been proved to significantly affect the asymptotic worst-case running time of push-relabel algorithms [2]: as a matter of fact, if active nodes are stored in a queue, the algorithm, usually referred to as the FIFO preflow-push algorithm, takes $O(n^3)$ in the worst case; if active nodes are kept in a priority queue where each extracted node has the maximum estimated distance from the sink, the worst-case running time decreases to $O(\sqrt{m}n^2)$, which is much better for sparse graphs. The last algorithm is known as the highest-level preflow-push algorithm.

Unfortunately, regardless of the selection strategy, the push-relabel method yields in practice very slow codes if taken “ad litteram”. Indeed, the way estimated distances from the sink are maintained has been proved to dramatically affect the practical performances of the push-relabel algorithms. For this reason, several additional heuristics for the problem have been proposed. Though these heuristics are irrelevant from an asymptotic point of view, the experimental study presented in [10] proves that two of them, i.e., the global relabeling and the gap heuristics, could be extremely useful in practice.

Global Relabeling Heuristic. Each relabel operation increases the estimated distance of the current active node from the sink to be equal to the lowest estimated distance of any adjacent node, plus one. This is done by considering only adjacent nodes joined by edges with some non-zero residual capacity, i.e., edges that can still carry some additional flows. As relabel operations are indeed local operations, the estimated distances from the sink may progressively deviate from the exact distances by losing the “big picture” of the distances: for this reason, flow excesses might not be correctly pushed right ahead towards the sink, and may follow longer paths slowing down the computation. The global relabeling heuristic consists of recomputing, say every n push/relabel operations, the exact distances from the sink, and the asymptotic cost of doing so can be amortized against the previous operations. This heuristic drastically improves the practical running time of algorithms based on the push-relabel method [10].

Gap Heuristic. Different authors [9] have observed that, at any time during the execution of the algorithm, if there are nodes with estimated distances from the sink that are strictly greater than some distance d and no other node has estimated distance d , then a gap in the distances has been formed and all active nodes above the gap will eventually send their flow excesses back to the source as they no more reach the sink. This is achieved by the algorithm through successive repeated increments of the estimated distances done via relabel operations until they get greater than n . The problem is that a huge number of such relabeling operations may be required. To avoid this, it is possible to efficiently keep track of gaps in the distances: whenever a gap occurs, the estimated distances of all nodes above the gap are immediately increased to n . This is usually referred to as the gap heuristic and, according to the study in [10], it is a very useful addition to the global relabeling heuristics if the highest-level active node selection strategy is applied. However, the gap heuristic does not seem to yield the same improvements under FIFO selection strategy.

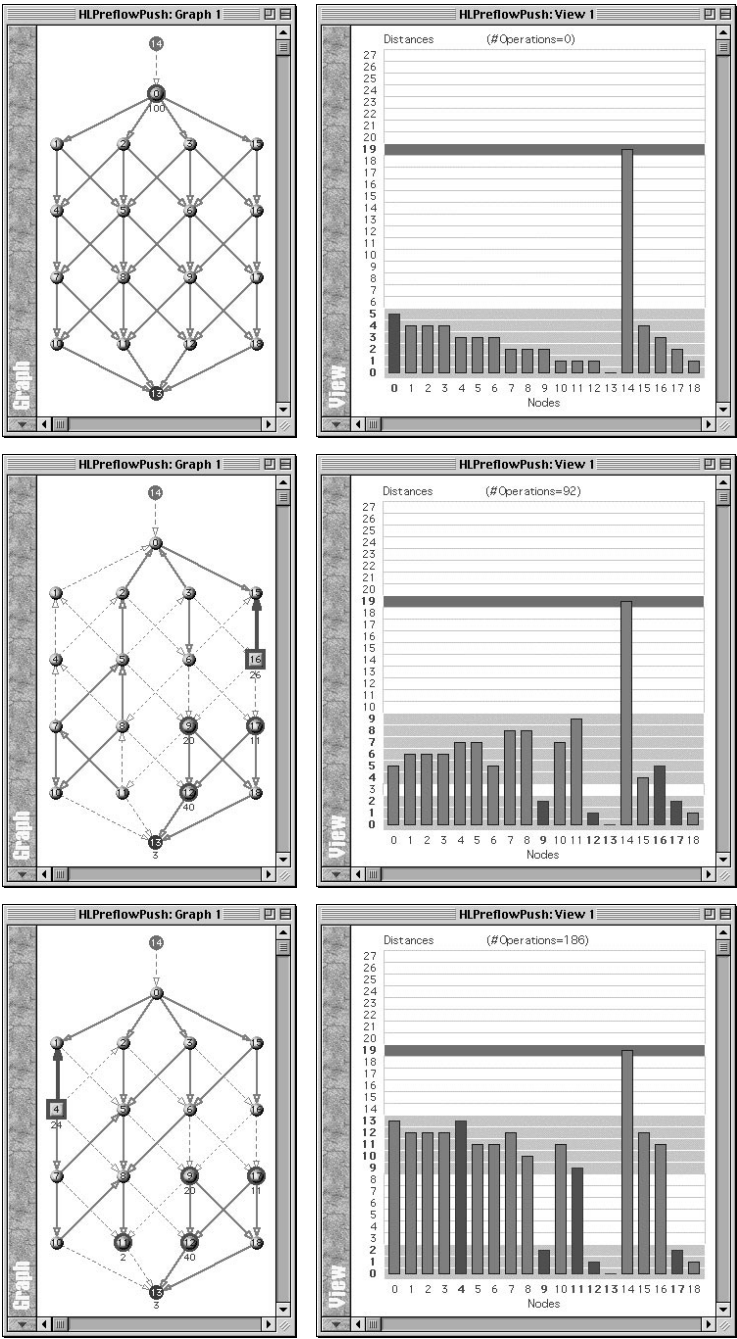


Fig. 1. Highest-level preflow push maxflow algorithm animation. Snapshots a, b, c.

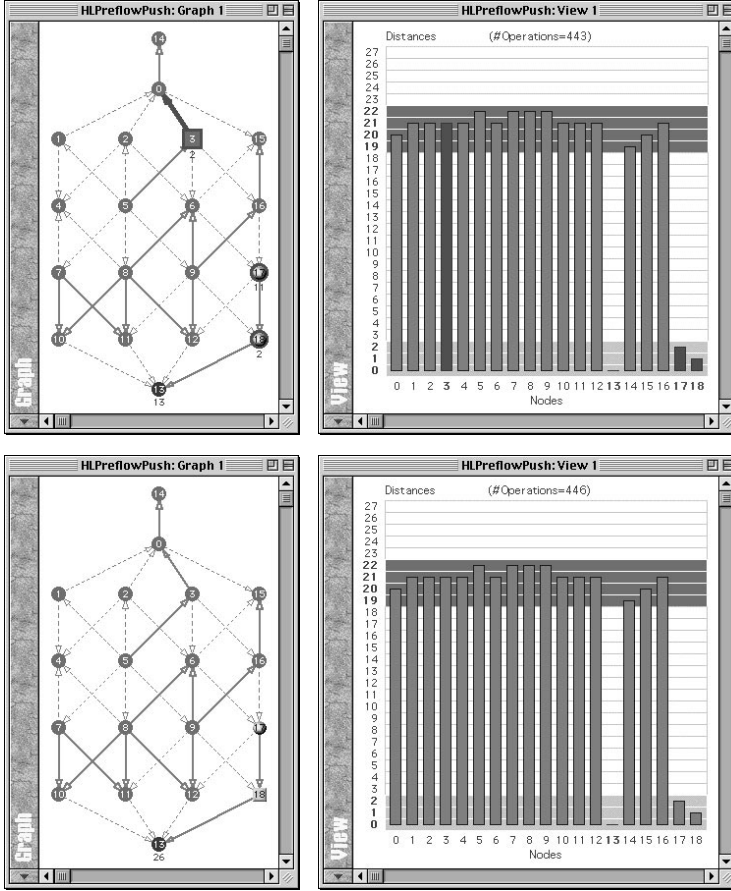


Fig. 2. Highest-level preflow push maxflow algorithm animation. Snapshots d, e.

The 5 snapshots *a*, *b*, *c*, *d* and *e* shown in Fig. 1 and in Fig. 2 have been produced by the algorithm animation system Leonardo [12] and depict the behavior of the highest-level preflow push algorithm implemented with no additional heuristics on a small network with 19 nodes and 39 edges. The animation aims at giving an empirical explanation about the utility of the gap heuristic under the highest-level selection. The example shows that this heuristic, if added to the code, could have saved about 80% of the total time spent by the algorithm to solve the problem on that instance. Both the network and a histogram of the estimated distances of nodes are shown in the snapshots: active nodes are highlighted both in the network and in the histogram and flow excesses are reported as node labels. Moreover, the edge currently selected for a push operation is highlighted as well. Notice that the source is initially assigned distance n and all nodes that eventually send flows back to the source get distance greater than n .

3.2 Matrix Multiplication

In this section we briefly report on an experimental study of matrix multiplication [5]. The study is an example of the fact that theoretical conclusions on locality exploitation can yield practical implementations with the desired properties. Differently from the case of Maxflow presented in Section 3.1 where experiments helped theoreticians develop suitable heuristics for improving the running time, experiments in this case provided an empirical confirmation about the precision of theoretical performance prediction related to certain matrix multiplication algorithms.

In general, as memory hierarchy is of no help to performance if the computation exhibits an insufficient amount of locality, both algorithm design and compiler optimizations should explicitly take into account locality.

For what concerns matrix multiplication, a simple approach, called *fractal* approach [5], allows it to design algorithms that expose locality at all temporal scales. The main idea consists of decomposing matrices recursively embedding any two-dimensional array X into a one-dimensional array Y as follows:

$$X = \begin{bmatrix} A_0 & A_1 \\ A_2 & A_3 \end{bmatrix} \qquad Y = \begin{bmatrix} A_0 & A_1 & A_2 & A_3 \end{bmatrix}$$

Keeping in mind this decomposition, it is possible to define a class of recursive algorithms that compute a product $C \leftarrow A \cdot B$ by performing 8 recursive computations on smaller matrices. Below we show two possible orderings for computing the multiplications of sub-matrices, corresponding to two different algorithms, namely **ABC-fractal** and **CAB-fractal**:

	ABC-fractal		CAB-fractal
1	$C_0 \leftarrow C_0 + A_0 \cdot B_0$	1	$C_0 \leftarrow C_0 + A_0 \cdot B_0$
2	$C_1 \leftarrow C_1 + A_0 \cdot B_1$	2	$C_0 \leftarrow C_0 + A_1 \cdot B_2$
3	$C_3 \leftarrow C_3 + A_2 \cdot B_1$	3	$C_1 \leftarrow C_1 + A_0 \cdot B_1$
4	$C_2 \leftarrow C_2 + A_2 \cdot B_0$	4	$C_1 \leftarrow C_1 + A_1 \cdot B_3$
5	$C_2 \leftarrow C_2 + A_3 \cdot B_2$	5	$C_3 \leftarrow C_3 + A_2 \cdot B_1$
6	$C_3 \leftarrow C_3 + A_3 \cdot B_3$	6	$C_3 \leftarrow C_3 + A_3 \cdot B_3$
7	$C_1 \leftarrow C_1 + A_1 \cdot B_3$	7	$C_2 \leftarrow C_2 + A_2 \cdot B_0$
8	$C_0 \leftarrow C_0 + A_1 \cdot B_2$	8	$C_2 \leftarrow C_2 + A_3 \cdot B_2$

From the perspective of temporal locality, there is always a sub-matrix in common between consecutive calls, which increases data reuse. In particular, it is not difficult to see that both **ABC-fractal** and **CAB-fractal** actually maximize data reuse. Moreover, the first algorithm optimizes *read* cache misses, while the second optimizes *write* cache misses. This is clear if we consider, for example, that sub-matrix sharing between consecutive calls in **CAB-fractal** is maximum for C , which is the matrix being written.

Experiments showed that these algorithms can efficiently exploit the cache hierarchy without taking cache parameters into account, thus ensuring portability of cache performance [5].

References

1. A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.
2. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.
3. R. Anderson. The role of experiment in the theory of algorithms. In *Proceedings of the 5th DIMACS Challenge Workshop*, 1996. Available over the Internet at the URL: <http://www.cs.amherst.edu/~dsj/methday.html>.
4. C. Berge and A. Ghouila-Houri. *Programming, Games and Transportation Networks*. Wiley, 1962.
5. G. Bilardi, P. D'Alberto, and A. Nicolau. Fractal matrix multiplication: a case study on portability of cache performance. Manuscript, May 2000.
6. M.H. Brown. *Algorithm Animation*. MIT Press, Cambridge, MA, 1988.
7. M.H. Brown. Zeus: a System for Algorithm Animation and Multi-View Editing. In *Proceedings of the 7-th IEEE Workshop on Visual Languages*, pages 4–9, 1991.
8. G. Cattaneo, U. Ferraro, G.F. Italiano, and V. Scarano. Cooperative Algorithm and Data Types Animation over the Net. In *Proc. XV IFIP World Computer Congress, Invited Lecture*, pages 63–80, 1998. System Home Page: <http://isis.dia.unisa.it/catai/>.
9. B.V. Cherkassky. *A Fast Algorithm for Computing Maximum Flow in a Network*. In A.V. Karzanov editor, *Collected Papers, Issue 3: Combinatorial Methods for Flow Problems*, pages 90–96. The Institute for Systems Studies, Moscow, 1979. In Russian. English translation appears in AMS Translations, Vol. 158, pp. 23–30. AMS, Providence, RI, 1994.
10. B.V. Cherkassky and A.V. Goldberg. On implementing the push-relabel method for the maximum flow problem. *Algorithmica*, 19:390–410, 1997.
11. T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.
12. P. Crescenzi, C. Demetrescu, I. Finocchi, and R. Petreschi. Reversible Execution and Visualization of Programs with LEONARDO. *Journal of Visual Languages and Computing*, 11(2), 2000. Leonardo is available at the URL: <http://www.dis.uniroma1.it/~demetres/Leonardo/>.
13. G.B. Dantzig. *Application of the Simplex Method to a Transportation Problem*. In T.C. Hoopmans editor, *Activity Analysis and Production and Allocation*, Wiley, New York, 1951.
14. Demetrescu, C. and Finocchi, I. Break the “Right” Cycles and Get the “Best” Drawing. In *Proc. of the 2nd International Conference on Algorithms and Experimentations (ALENEX'00)*, San Francisco, CA, 2000.
15. M. Eisenstadt and M. Brayshaw. The transparent prolog machine: An execution model and graphical debugger for logic programming. *Journal of Logic Programming*, 5(4):1–66, 1988.
16. A.V. Goldberg. Selecting problems for algorithm evaluation. In *Proc. 3-rd Workshop on Algorithm Engineering (WAE'99)*, LNCS 1668, pages 1–11, 1999.
17. A.V. Goldberg and B.M.E. Moret. Combinatorial algorithms test sets [CATS]: The ACM/EATCS platform for experimental research (short). In *SODA: ACM-SIAM Symposium on Discrete Algorithms*, 1999.
18. R.R. Henry, K.M. Whaley, and B. Forstall. The University of Washington Program Illustrator. In *Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation*, pages 223–233, 1990.

19. D. Johnson. A theoretician's guide to the experimental analysis of algorithms. In *Proceedings of the 5th DIMACS Challenge Workshop*, 1996. Available over the Internet at the URL: <http://www.cs.amherst.edu/dsj/methday.html>.
20. D. Klingman, A. Napier, and J. Stutz. Netgen: A program for generating large scale capacitated assignment, transportation, and minimum cost network flow problems. *Management Science*, 20:814–821, 1974.
21. Donald E. Knuth. Stanford GraphBase: A platform for combinatorial algorithms. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 41–43, New York, NY 10036, USA, 1993. ACM Press.
22. S.P. Lahtinen, E. Sutinen, and J. Tarhio. Automated Animation of Algorithms with Eliot. *Journal of Visual Languages and Computing*, 9:337–349, 1998.
23. T. Leong, P. Shor, and C. Stein. Implementation of a combinatorial multicommodity flow algorithm. In D.S. Johnson and C.C. McGeoch, eds., *Network Flows and Matching: First DIMACS Implementation Challenge*, pages 387–406, 1993.
24. C. McGeoch. A bibliography of algorithm experimentation. In *Proceedings of the 5th DIMACS Challenge Workshop*, 1996. Available over the Internet at the URL: <http://www.cs.amherst.edu/dsj/methday.html>.
25. K. Mehlhorn and S. Naher. LEDA, a platform for combinatorial and geometric computing. *Communications of the ACM*, 38:96–102, 1995.
26. B.M.E. Moret. Towards a discipline of experimental algorithmics. In *Proceedings of the 5th DIMACS Challenge Workshop*, 1996. Available over the Internet at the URL: <http://www.cs.amherst.edu/dsj/methday.html>.
27. B.M.E. Moret and H.D. Shapiro. An empirical assessment of algorithms for constructing a minimal spanning tree. *Computational Support for Discrete Mathematics* N. Dean and G. Shannon eds. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 15:99–117, 1994.
28. G.C. Roman, K.C. Cox, C.D. Wilcox, and J.Y. Plun. PAVANE: a System for Declarative Visualization of Concurrent Computations. *Journal of Visual Languages and Computing*, 3:161–193, 1992.
29. S. Skiena. Who is interested in algorithms and why? lessons from the stony brook algorithms repository. In *Proc. Workshop on Algorithm Engineering (WAE'98)*, pages 204–212, 1998.
30. J.T. Stasko. The Path-Transition Paradigm: a Practical Methodology for Adding Animation to Program Interfaces. *Journal of Visual Languages and Computing*, 1(3):213–236, 1990.
31. J.T. Stasko. A Methodology for Building Application-Specific Visualizations of Parallel Programs. *Journal of Parallel and Distributed Computing*, 18:258–264, 1993.
32. J.T. Stasko, J. Domingue, M.H. Brown, and B.A. Price. *Software Visualization: Programming as a Multimedia Experience*. MIT Press, Cambridge, MA, 1997.

And/Or Hierarchies and Round Abstraction

Radu Grosu

Department of Computer and Information Science
University of Pennsylvania
Email: grosu@cis.upenn.edu
URL: www.cis.upenn.edu/~grosu

Abstract. Sequential and parallel composition are the most fundamental operators for incremental construction of complex concurrent systems. They reflect the temporal and respectively the spatial properties of these systems. Hiding temporal detail like internal computation steps supports temporal scalability and may turn an asynchronous system to a synchronous one. Hiding spatial detail like internal variables supports spatial scalability and may turn a synchronous system to an asynchronous one. In this paper we show on hand of several examples that a language explicitly supporting both sequential and parallel composition operators is a natural setting for designing heterogeneous synchronous and asynchronous systems. The language we use is SHRM, a visual language that backs up the popular and/or hierarchies of statecharts with a well defined compositional semantics.

1 Introduction

With the advent of very large scale integration technology (VLSI), digital circuits became too complex to be designed and tested on a breadboard. The hardware community introduced therefore languages like Verilog and VHDL [Ver,Vhdl] that allow to describe the architectural and the behavioral structure of a complex circuit in a very abstract and modular way. Architectural modularity means that a system is composed of subsystems using the operations of parallel composition and hiding of variables. Behavioral hierarchy means that a system is composed of subsystems using the operations of sequential composition and hiding of internal computation steps. Verilog allows the arbitrary nesting of the architecture and behavior hierarchies.

With the advent of object oriented technology, most notably UML [BJR97], combined visual/textual languages very similar in spirit to the hardware description languages [Har87,SGW94], gained a lot of popularity in the software community. Their behavior and block diagrams were rapidly adopted as a high level interface for Verilog and VHDL too (e.g. in the Renoir tool of Mentor Graphics and in the StateCad tool of Visual Software Solutions).

Recent advances in formal verification have led to powerful design tools for hardware (see [CK96] for a survey), and subsequently, have brought a lot of hope of their application to reactive programming. The most successful verification technique has been *model checking* [CE81,QS82]. In model checking, the

system is described by a state-machine model, and is analyzed by an algorithm that explores the reachable state-space of the model. The state-of-the-art model checkers (e.g. SPIN [Hol97] and SMV [McM93]) employ a variety of heuristics for efficient search, but are typically unable to analyze models with more than hundred state variables, and thus, *scalability* still remains a challenge.

A promising approach to address scalability is to exploit the modularity of the design. The input languages of standard model checkers (e.g., S/R in Cospan [AKS83] or Reactive modules in Mocha [AH99]) support architectural modularity but, unlike the hardware and the visual description languages, provide no support for modular description of the behaviors of individual components. In [AG00] we introduced the combined visual/textual language *hierarchic reactive modules* (HRM) exhibiting both behavior and architecture modularity. This hierarchy is exploited for efficient search by the model checker HERMES [AGM00].

In this paper we introduce a synchronous version of the hierarchic reactive modules language (SHRM) that conservatively extends the reactive modules language. This language is used to model two very interesting abstraction operators of reactive modules: **next** and its dual **trigger**. They allow to collapse and delay arbitrary many consecutive steps of a module and environment respectively, and therefore to perform a temporal abstraction. This abstraction can be exploited efficiently in model checking because the states stored for the intermediate steps may be discarded. We argue that a language like SHRM and Verilog, supporting the arbitrary nesting of architecture and behavior hierarchies, is a natural setting for combined spatial and temporal abstraction. There is no need for special temporal operators because behavioral modularity does precisely the same thing. Moreover, by supporting sequential composition, choice, loops and pre-emption constructs, the combined setting allows to express complex structure in a more direct and intuitive way. To materialize this claim we reformulate the adder example in [AH99].

The rest of the paper is organized as follows. In Section 2 we introduce the modeling language SHRM. This language adds communication by events to the language HRM presented in [AG00]. It also extends the reactive modules language both with behavior hierarchy and with a visual notation. In Section 3 we show that this language is a natural setting to perform spatial and temporal abstraction. As an application, we show how to encode the next operator of reactive modules. Finally in Section 4 we draw some conclusions.

2 Modeling Language

The central component of the modeling language is a *mode*. The attributes of a mode include global variables used to share data with its environment, local variables, well-defined entry and exit points, and submodes that are connected with each other by transitions. The transitions are labeled with guarded commands that access the variables according to the the natural scoping rules. Note that the transitions can connect to a mode only at its entry/exit points, as in

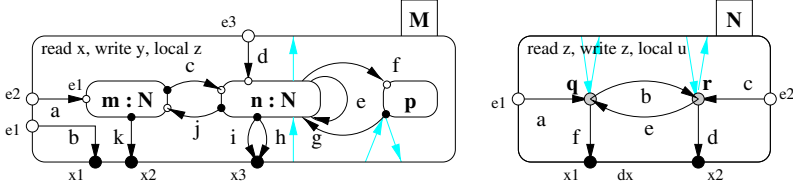


Fig. 1. Mode diagrams

ROOM but unlike STATECHARTS. This choice is important in viewing the mode as a black box whose internal structure is not visible from outside. The mode has a *default exit* point, and transitions leaving the default exit are applicable at all control points within the mode and its submodes. The default exit retains the history, and the state upon exit is automatically restored by transitions entering the default *entry point*. Thus, a transition from the default exit is a group preemption transition and a transition from the default exit to the default entry is an group interrupt transition. While defining the operational semantics of modes, we follow the standard paradigm in which transitions are executed repeatedly until there are no more enabled transitions.

Modes. A *mode* has a refined control structure given by a hierarchical state machine. It basically consists of a set of *submode instances* connected by *transitions* such that at each moment of time *only one* of the submode instances *is active*. A submode instance has an associated mode and we require that the modes form an *acyclic* graph with respect to this association. For example, the mode M in Figure 1 contains two submode instances, m and n pointing to the mode N. By distinguishing between modes and instances we may control the degree of *sharing* of submodes. Sharing is highly desirable because submode instances (on the same hierarchy level) are never simultaneously active in a mode. Note that a mode resembles an *or* state in STATECHARTS but it has more powerful structuring mechanisms.

Variables and Scoping. A mode may have *global* as well as *local variables*. The set of global variables is used to share data with the mode's environment. The global variables are classified into *read* and *write variables*. The *local variables* of a mode are accessible only by its transitions and submodes. The local and write variables are called *controlled variables*. Thus, the scoping rules for variables are as in standard structured programming languages. For example, the mode M in Figure 1 has the global read variable x , the global write variable y and the local variable z . Similarly, the mode N has the global read-write variable z and the local variable u . Each variable x may be used as a register. In this case, the expression $p(x)$ denotes the value of x in the *previous top level round*¹ and the expression x denotes the *current* value of x .

¹ What previous top level round means will be made clear when discussing parallel modes.

The transitions of a mode may refer only to the declared global and local variables of that mode and only according to the declared read/write permission. For example, the transitions $a, b, c, d, e, f, g, h, i, j$ and k of the mode M may refer only to the variables x , y and z . Moreover, they may read only x and z and write y and z . The global and local variables of a mode may be shared between submode instances if the associated submodes declare them as global (the set of global variables of a submode has to be included in the set of global and local variables of its parent mode). For example, the value of the variable z in Figure 1 is shared between the submode instances m and n . However, the value of the local variable u is not shared between m and n .

Control Points and Transitions. To obtain a modular language, we require the modes to have well defined *control points* classified into entry points (marked as white bullets) and exit points (marked as black bullets). For example, the mode M in Figure 1 has the entry points $e1, e2, e3$ and the exit points $x1, x2, x3$. Similarly, the mode N has the entry points $e1, e2$ and the exit points $x1, x2$. The transitions connect the control points of a mode and of its submode instances to each other. For example, in Figure 1 the transition a connects the entry point $e2$ of the mode M with the entry point $e1$ of the submode instance m . The name of the control points of a transition are attributes and our drawing tool allows to optionally show or hide them to avoid cluttering.

According to the points they connect, we classify the transitions into *entry*, *internal* and *exit* transitions. For example, in Figure 1, a, d are *entry* transitions, h, i, k are *exit* transitions, b is an *entry/exit* transition and c, e, f, g, j are *internal* transitions. These transitions have different types. Entry transitions initialize the controlled variables by reading only the global variables. Exit transitions read the global and local variables and write only the global variables. The internal transitions read the global and the local variables and write the controlled variables.

Default Control Points. To model preemption each mode (instance) has a special, default exit point dx . In mode diagrams, we distinguish the default exit point of a mode from the regular exit points of the mode, by considering the default exit point to be represented by the mode's border. A transition starting at dx is called a *preempting* or *group transition* of the corresponding mode. It may be taken whenever the control is inside the mode and no internal transition is enabled. For example, in Figure 1, the transition f is a group transition for the submode n . If the current control point is q inside the submode instance n and neither the transition b nor the transition f is enabled, then the control is transferred to the default exit point dx . If one of e or f is enabled and taken then it acts as a preemption for n . Hence, inner transitions have a higher priority than the group transitions, i.e., we use *weak preemption*. This priority scheme facilitates a modular semantics. As shown in Figure 1, the transfer of control to the default exit point may be understood as a *default exit transition* from an exit point x of a submode to the default exit point dx that is enabled if and only if, all the explicit outgoing transitions from x are disabled. We exploit this intuition in the symbolic checker.

History and Closure. To allow history retention, we use a special default entry point de . As with the default exit points, in mode diagrams the default entry point of a mode is considered to be represented by the mode's border. A transition entering the default entry point of a mode either restores the values of all local variables along with the position of the control or initializes the controlled variables according to the read variables. The choice depends on whether the last exit from the mode was along the default exit point or not. This information is implicitly stored in the constructor of the state passed along the default entry point. For example, both transitions e and g in Figure 1, enter the default entry point de of n . The transition e is called a *self* group transition. A self group transition like e or more generally a self loop like f, p, g may be understood as an interrupt handling routine. While a self loop may be arbitrarily complex, a self transition may do simple things like counting the number of occurrences of an event (e.g., clock events). Again, the transfer of control from the default entry point de of a mode to one of its internal points x may be understood as a *default entry transition* that is taken when the value of the local history variable coincides with x . If x was a default exit point $n.dx$ of a submode n then, as shown in Figure 1, the default entry transition is directed to $n.de$. The reason is that in this case, the control was blocked somewhere inside of n and default entry transitions originating in $n.de$ will restore this control. A mode with added default entry and exit transitions is called *closed*. Note that the closure is a semantic concept. The user is not required to draw the implicit default entry and exit transitions. Moreover, he can override the defaults by defining explicit transitions from and to the default entry and exit points.

Operational Semantics: Macro-Steps. In Figure 1, the execution of a mode, say n , starts when the environment transfers the control to one of its entry points $e1$ or $e2$. The execution of n terminates either by transferring the control back to the environment along the exit points $x1$ or $x2$ or by “getting stuck” in q or r as all transitions starting from these leaf modes are disabled. In this case the control is implicitly transferred to M along the default exit point $n.dx$. Then, if the transitions e and f are enabled, one of them is nondeterministically chosen and the execution continues with n and respectively with p . If both transitions are disabled the execution of M terminates by passing the control implicitly to its environment at the default exit $M.dx$. Thus, the transitions within a mode have a higher priority compared to the group transitions of the enclosing modes.

Intuitively, a round of the machine associated to a mode starts when the environment passes the updated state along a mode's entry point and ends when the state is passed to the environment along a mode's exit point. All the internal steps (the *micro steps*) are hidden. We call a round also a *macro step*. Note that the macro step of a mode is obtained by alternating its closed transitions and the macro steps of the submodes.

Denotational Semantics: Traces. The execution of a mode may be best understood as a game, i.e., as an alternation of moves, between the mode and its environment. In a *mode move*, the mode gets the state from the environment

along its entry points. It then keeps executing until it gives the state back to the environment along one of its exit points. In an *environment move*, the environment gets the state along one of the mode's exit points. Then it may update any variable except the mode's local ones. Finally, it gives the state back to the mode along one of its entry points. An *execution* of a mode M is a sequence of macro steps of the mode. Given such an execution, the corresponding *trace* is obtained by projecting the states in the execution to the set of global variables. The *denotational semantics* of a mode M consists of its control points, global variables, and the set of its traces.

Atoms and Parallel Modes. An *atom* is a mode having only two points, the default entry point and the default exit point. A *parallel mode* is a very convenient abbreviation for a particular mode consisting of the parallel composition of atoms. To avoid race conditions, the parallel composition of atoms is defined only if (1) the atoms write disjoint sets of variables and (2) there is no cyclic dependency among the variables of different atoms (this similar to [AH99] and it can be statically checked). A weaker form of cyclic dependency is however allowed: for any write variable x in an atom A , another atom B may safely refer to $p(x)$, the *previous value* of x . If the atom B refers to x , then it refers to the last value of x , i.e., the value of x produced at the end of the subround of A . The atom B has therefore to *await* the atom A .

Since a mode may update a controlled variable x several times, we have to make sure that $p(x)$ is well defined, no matter how many times the variable is updated. In the following, we consider $p(x)$ to be the value of x at the *end of the previous top level round*. A top level round is the round of the *top level atom* containing x . Syntactically, a top level atom is an atom prefixed by the keyword **top**. Semantically, a top level atom makes sure that at the end of each round, $p(x)$ is updated to the current value of x .

Top level atoms fix the granularity of interaction and therefore they may be used only in the parallel composition of other top level atoms (parallel composition does not alter this granularity). Modes and parallel modes also fix the spatial and temporal granularity of computation. Modes and top level atoms in SHRM closely correspond to tasks and modules in Verilog. Tasks are programming units whereas modules are simulation units.

Semantics of Parallel Modes. The semantics of a parallel mode is very similar to the semantics of modules in [AH99]. As shown in [AG00] this semantics can be completely defined in terms of modes as follows. Take an arbitrary linearization of the await dependency among the atoms of a parallel mode (since the await dependency is a partial order, this is always possible). Construct a mode by connecting the atoms with identity transitions, as required by the linearization. If the parallel mode is a top level atom, update $p(x)$ to the last value of x^2 . The language generated by this mode, defines the semantics of the parallel mode.

² This is a simpler definition than in [AG00]. This is because we use here the notion of top level atoms.

By definition, a parallel mode is a particular atom. As a consequence it may be freely used inside a mode as a submode. Hence, SHRM allows the arbitrary nesting of the architecture and behavior hierarchies. When conveniently, we will draw a parallel mode as a block diagram with atoms as boxes and shared variables as arrows. The entry/exit point information is not very informative for parallel modes (and atoms).

Events. The shared variables communication paradigm and the notion of top level round allows us to model *events* as toggling boolean variables. Sending an event e is the action $e := \neg p(e)$ and receiving an event e is the boolean expression $e \neq p(e)$. These are abbreviated by $e!$ and $e?$ respectively. Note that, no matter how many times a mode sends an event inside a top level round, only one event is sent to the other modes.

Renaming of Modes. Similarly to modules in [AH99], modes may be renamed. Given a mode m and a renaming $x_1, \dots, x_n := y_1, \dots, y_n$ where x_i are global variables and y_i are fresh variables, the mode $m[x_1, \dots, x_n := y_1, \dots, y_n]$ is a mode identical with m excepting that the variables x_i are replaced with the variables y_i , for $1 \leq i \leq n$.

3 Temporal and Spatial Abstraction

In order to reduce the complexity of a system, [AH99] introduce the abstraction operator **next**. Given a module m and a subset Y of its interface (write) variables, **next** Y **for** m collapses consecutive rounds of m until one of the variables in Y changes its value.

A *controlled state* of m is a valuation for the controlled variables of m and an *external state* of m is a valuation for the external (read) variables of m . For two external states s and t of m , an *iteration of m from s to t* is a finite sequence $s_0 \dots s_n$ of controlled states of m such that $n \geq 1$ and for all $0 \leq i < n$ the state $s_{i+1} \cup t$ is an successor of the state $s_i \cup s$. In other words, along an iteration the controlled variables are updated while the external variables stay unchanged.

The iteration $s_0 \dots s_n$ *modifies* the set Y of controlled variables if $s_n[Y] \neq s_0[Y]$ and for all $0 \leq i < n$, $s_i[Y] = s_0[Y]$, where $s[Y]$ is the projection of the state s on the variables in Y . If the iteration modifies Y then the state $s_n \cup t$ is called the *Y -successor* of the state $s_0 \cup s$. A *round marker* for the module m is a nonempty set Y of interface variables such that for all states s and t of m , there are nonzero and finitely many Y -successors u of s such that u and t agree on the values of the external (read) variables of m .

If Y is a *round marker* for the module m , then the abstraction **next** Y **for** m is a module with the same declaration as m and a single atom A_m^Y . The update relation of A_m^Y contains pairs (s, t) where t is a Y -successor of s .

Within the language SHRM, the next abstraction is a simple but important case of sequential control on top of parallel modes. Given a (parallel) mode m and a round marker Y , the mode corresponding to **next** Y **for** m is shown in

Figure 2. The game semantics of modes provides exactly the meaning of **next** above. The state (token) s is passed by the environment to the mode **next** Y

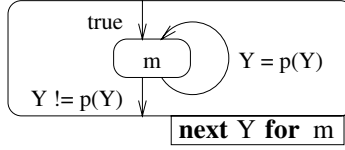


Fig. 2. Next abstraction

for m along its default entry point de . The state t is passed back by the mode to the environment along its default exit point dx only if t is a Y -successor of s (in this case $Y \neq p(Y)$). As long as the state token is inside **next** the environment does not have any chance to modify it. As a consequence, the states $s_0 \dots s_{n-1}$ computed by repeatedly traversing the loop are an iteration for this mode. None of these states are Y -successors of s because of the loop guard $Y = p(Y)$. Since the set Y is a round marker for m there is always the possibility for the loop to terminate. The textual variant of Figure 2 is shown below³.

```

atom next (m, Y) is
  read m.read;
  write m.write;
  submode m;

  transition from de to m.de is
    true -> skip;
  transition from m.dx to m.de is
    Y = p(Y) -> skip;
  transition from m.dx to dx is
    Y != p(Y) -> skip;

```

A generalization of the next operation above is the *reuse* of a (parallel) mode. In this case additional control is needed to prepare the input and store the output of the reused mode. For example, consider a one bit *adder* implemented as a parallel mode as shown in Figure 3.

³ The mode m and the set Y are considered parameters in this specification. The selectors $m.read$ and $m.write$ return the read and write variables of m .

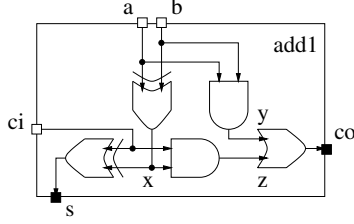


Fig. 3. One bit adder

Its textual equivalent is given below. It reproduces the circuit in Figure 3.

```

atom add1 is
  read a,b,ci : bool;
  write s, co : bool;
  local x, y, z : bool;

  || xor[in1, in2, out := a, b, x]
  || and[in1, in2, out := a, b, y]
  || xor[in1, in2, out := ci, x, s]
  || and[in1, in2, out := ci, x, z]
  || or[in1, in2, out := y, z, co]

```

Suppose now that we want to define a two bit adder by using in parallel two one bit adders, i.e., by decomposing the two bit addition *spatially*. The *spatial scaling involves a local variable (wire)*, that passes the the carry bit from the lower bit adder to the higher bit adder. Hence, spatial abstraction involves hiding of local variables as shown in Figure 4 left. The textual equivalent is given below. Note that the spatial abstraction does not change the notion of a round (or clock cycle). This remains the same for all modes (circuits) constructed in this way. Combinational cycles are prohibited by the parallel composition operation.

```

atom pAdd2 is
  read x, y : array (0..1) of bool; cIn : bool;
  write z : array (0..1) of bool; cOut : bool;
  local c : bool;

  || add1[a, b, s, ci, co := x[0], y[0], z[0], cIn, c]
  || add1[a, b, s, ci, co := x[1], y[1], z[1], c, cOut]

```

Suppose now that we want to define the two bit adder by reusing the one bit adder, i.e., by decomposing the two bit addition *temporally*. This implementation splits each computation step into two micro-steps. In the first micro-step the one bit adder is used to add the lower bits. In the second micro-step the one bit adder is used to add the higher order bits. Similarly, an n -bit adder can be implemented in n micro-steps. To capture the *micro step* intuition, we have to *hide (or compress) the micro steps* into one computation step. But this exactly what mode encapsulation is about.

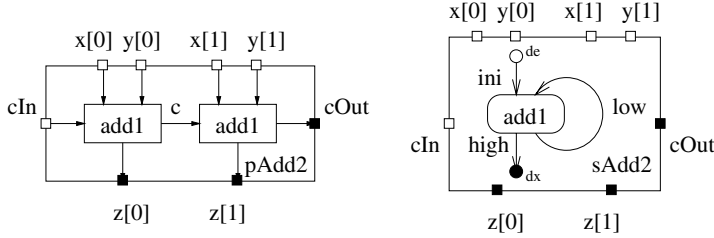


Fig. 4. Two bit adders

In contrast to the simple next operation defined before, in this case we also have to prepare the input for the one bit adder and to store the partial results. We also need a local counter to count the number of micro steps. This implementation is shown visually in Figure 4 right. Its textual definition is given below. The reader is urged to compare it with the less intuitive and much more involved implementation given in [AH99].

atom sAdd2 is

```

read x, y : array (0..1) of bool; cIn : bool;
write z : array (0..1) of bool; cOut : bool;
local a, b, s, ci, co, r : bool;

transition ini from de to add1.de is
  true -> r := 0; a := x[0]; b := y[0]; ci := cIn;
transition low from add1.dx to add1.de is
  r = 0 -> r := 1; z[0] := s; a := x[1]; b := y[1]; ci := co;
transition high from add1.dx to dx is
  r = 1 -> z[1] := s; cOut := co;
```

The game semantics of modes makes the **trigger** construct from [AH99] superfluous. As long as the top level atom does not pass the state token, the environment cannot modify it. As a consequence, it cannot work faster than the atom itself and this is exactly the purpose of **trigger**.

4 Conclusions

In this paper we have introduced a synchronous visual/textual modeling language for reactive systems that allows the arbitrary nesting of architectural and behavioral hierarchy. We have shown that such a language is the natural setting for spatial and temporal scaling and consequently for the modeling of heterogeneous synchronous and asynchronous (stuttering) systems. This language is more expressive than reactive modules because it allows to define behavior hierarchy. It is more expressive than hierarchic reactive modules because it supports communication by events. In a nutshell, it has much of the expressive power of Verilog and VHDL and yet it has a formal semantics that supports the efficient application of formal verification techniques, especially of model checking.

The additional expressive power with respect to reactive and hierarchic reactive modules does not come however, for free. When applying symbolic search (e.g. invariant checking) we have to introduce an additional fresh variable px for each each variable x addressed as $p(x)$. To avoid this waste, we could classify the variables like in VHDL, into proper variables and signals and disallow the repeated updating of signals. By insisting that only signals x can be addressed as $p(x)$ no additional space is required. In conclusion, even though experimental data is small so far, conceptual evidence suggests that a language supporting the arbitrary nesting of behavior and architecture hierarchy could be beneficial both for modeling and for analysis.

Acknowledgments. We would like to thank Rajeev Alur for reading a draft of this paper and providing valuable feedback. We would also like to thank Tom Henzinger for fruitful discussions and enthusiasm for a language supporting both hierarchies. This work was supported by the DARPA/NASA grant NAG2-1214.

References

- [AG00] R. Alur and R. Grosu. Modular refinement of hierarchic reactive machines. In *Proceedings of the 27th Annual ACM Symposium on Principles of Programming Languages*, pages 390–402, 2000.
- [AGM00] R. Alur, R. Grosu, M. McDougall. Efficient Reachability Analysis of Hierarchical Reactive Machines. In *Proceedings of the 12th Conference on Computer Aided Verification*, Chicago, USA, 2000.
- [AH99] R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.
- [AHM⁺98] R. Alur, T. Henzinger, F. Mang, S. Qadeer, S. Rajamani, and S. Tasiran. MOCHA: Modularity in model checking. In *Proceedings of the 10th International Conference on Computer Aided Verification*, LNCS 1427, pages 516–520. Springer-Verlag, 1998.
- [AKS83] S. Aggarwal, R.P. Kurshan, and D. Sharma. A language for the specification and analysis of protocols. In *IFIP Protocol Specification, Testing, and Verification III*, pages 35–50, 1983.
- [AKY99] R. Alur, S. Kannan, and M. Yannakakis. Communicating hierarchical state machines. In *Automata, Languages and Programming, 26th International Colloquium*, pages 169–178. 1999.
- [AY98] R. Alur and M. Yannakakis. Model checking of hierarchical state machines. In *Proceedings of the Sixth ACM Symposium on Foundations of Software Engineering*, pages 175–188. 1998.
- [BHSV⁺96] R. Brayton, G. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, S. Qadeer, R. Ranjan, S. Sarwary, T. Shiple, G. Swamy, and T. Villa. VIS: A system for verification and synthesis. In *Proceedings of the Eighth Conference on Computer Aided Verification*, LNCS 1102, pages 428–432. 1996.
- [BJR97] G. Booch, I. Jacobson, and J. Rumbaugh. *Unified Modeling Language User Guide*. Addison Wesley, 1997.
- [BLA⁺99] G. Behrmann, K. Larsen, H. Andersen, H. Hulgaard, and J. Lind-Nielsen. Verification of hierarchical state/event systems using reusability and compositionality. In *TACAS '99: Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Software*, 1999.

- [CAB⁺98] W. Chan, R. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering*, 24(7):498–519, 1998.
- [CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pages 52–71. Springer-Verlag, 1981.
- [CK96] E.M. Clarke and R.P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.
- [Har87] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
- [Hol91] G.J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.
- [Hol97] G.J. Holzmann. The model checker SPIN. *IEEE Trans. on Software Engineering*, 23(5):279–295, 1997.
- [JM87] F. Jahanian and A.K. Mok. A graph-theoretic approach for timing analysis and its implementation. *IEEE Transactions on Computers*, C-36(8):961–975, 1987.
- [LHHR94] N.G. Leveson, M. Heimdahl, H. Hildreth, and J.D. Reese. Requirements specification for process control systems. *IEEE Transactions on Software Engineerings*, 20(9), 1994.
- [McM93] K. McMillan. *Symbolic model checking: an approach to the state explosion problem*. Kluwer Academic Publishers, 1993.
- [PD96] L. Peterson and B. Davie. *Computer Networks: A Systems Approach*. Morgan Kaufmann, 1996.
- [Pet81] G. Peterson. Myths about the mutual exclusion problem. *Information Processing Letters*, 12(3), 1981.
- [QS82] J.P. Queille and J. Sifakis. Specification and verification of concurrent programs in CESAR. In *Proceedings of the Fifth International Symposium on Programming*, LNCS 137, pages 195–220. Springer-Verlag, 1982.
- [SGW94] B. Selic, G. Gullekson, and P.T. Ward. *Real-time object oriented modeling and design*. J. Wiley, 1994.
- [Ver] IEEE Standard 1364-1995. Verilog Hardware Description Language Reference Manual, 1995.
- [Vhdl] IEEE Standard 1076-1993. VHDL Language Reference Manual, 1993.

Computational Politics: Electoral Systems^{*}

Edith Hemaspaandra¹ and Lane A. Hemaspaandra²

¹ Department of Computer Science,
Rochester Institute of Technology, Rochester, NY 14623, USA

² Department of Computer Science, University of Rochester
Rochester, NY 14627, USA

Abstract. This paper discusses three computation-related results in the study of electoral systems:

1. Determining the winner in Lewis Carroll's 1876 electoral system is complete for parallel access to NP [22].
2. For any electoral system that is neutral, consistent, and Condorcet, determining the winner is complete for parallel access to NP [21].
3. For each census in US history, a simulated annealing algorithm yields provably fairer (in a mathematically rigorous sense) congressional apportionments than any of the classic algorithms—even the algorithm currently used in the United States [24].

1 Introduction

Political scientists have a number of natural properties that every electoral system arguably should, ideally, obey. Things are bad. There are quite reasonable, modest property lists such that it is known that no system can satisfy all the list's properties (see, e.g., [2]). Things are worse. Typically, computational feasibility isn't even on the list of properties.

To the computer scientist, this is troubling. After all, even if an election method has various natural, desirable properties from the point of view of political science, if it is computationally intractable then it probably should best be viewed as a nonstarter.

In fact, one can trace the origins of sensitivity to computational limitations on economic and political choice back many decades—for example, to Simon's insightful notion of bounded rationality ([38], see also [33]). And in economics, political science, computer science, and operations research, there has been extensive research on the effect of computational resource limitations on decision makers/players in games. However, in this article we are solely concerned with computational and tractability issues as they relate to *electoral (voting) systems*. On this topic, a decade ago, a set of extremely perceptive, provocative papers by Bartholdi, Tovey, and Trick explored this direction, proved lower bounds, and stated challenging issues [6,5,7].

^{*} Email: eh@cs.rit.edu, lane@cs.rochester.edu. Supported in part by grant NSF-INT-9815095/DAAD-315-PPP-gü-ab. Work done in part while visiting Julius-Maximilians-Universität Würzburg.

Recently, many of these lower bounds have been significantly raised, and matching upper bounds provided—obtaining exact classification of many problems of electoral evaluation. Sections 2 and 3 present some problems studied and the results obtained. We will see that some attractive electoral systems have the flaw that the problem of determining who won is of extraordinarily high complexity—typically, complete for parallel access to NP.

Section 4 looks at experimental work aimed at understanding and counteracting the biases built into the apportionment process—for example, of the US Congress. Remarkably, many of the greatest figures in American history—Thomas Jefferson, John Quincy Adams, Alexander Hamilton, and Daniel Webster—designed apportionment algorithms and debated the algorithms’ merits. The debate they started has raged for more than 200 years—in the year 1792 an apportionment bill caused President George Washington to cast the first veto in US history, and yet in the 1990s the Supreme Court was still weighing what degree of flexibility Congress has in selecting apportionment algorithms [41]. However, a new, mathematical view of power and fairness developed in the 1900s, when viewed in light of experimental algorithmics, has opened new possibilities, and has led to proofs that current and past apportionments are unfair.

In Sections 2 through 4, our primary goal is to describe the electoral systems, to describe the interesting problems that the particular electoral issue poses, to discuss what insights into the computation or complexity of the problems have been obtained, and to comment on what the computational or complexity-theoretic insights say about the wisdom or attractiveness of the electoral system. So, we will state (with references) what complexity-theoretic results have been obtained, without here reproving such results.

2 Lewis Carroll’s Election System: Telling Who Won Is $P_{||}^{NP}$ -Complete

In the late 1700s, Marie-Jean-Antoine-Nicolas Caritat, the Marquis de Condorcet, noticed a troubling feature of majority-rule democracy: Even if each voter has rational preferences (i.e., has no strict cycles in his or her preferences), society’s aggregate preferences under pairwise majority-rule comparisons may be irrational [13]. For example, consider the following hypothetical election: 50,000,000 voters like Pat Buchanan least, Al Gore more, and George W. Bush most; 40,000,000 voters like George W. Bush least, Pat Buchanan more, and Al Gore most; and 30,000,000 voters like Al Gore least, George W. Bush more, and Pat Buchanan most. So, in pairwise comparisons, Pat loses to Al (by 60,000,000 votes) and Al loses to George (by 40,000,000 votes), yet George loses to Pat (by 20,000,000 votes)! Society has a strict cycle in its preferences:

$$Al < George < Pat < Al.$$

The fact that a society of individually rational people can be irrational when aggregated under pairwise majority-rule contests is known as the *Condorcet Paradox*. Of course, the Condorcet Paradox is not a paradox—it is just a feature.

Lewis Carroll (whose real name was Charles Lutwidge Dodgson), the Oxford mathematics professor and author, in the 1800s noticed the Condorcet Paradox, though probably independently of Condorcet (see the discussion in [9]). Carroll developed a fascinating electoral system that was guaranteed never to aggregate rational voters into an irrational societal view [14].

We now describe Carroll's system (for other descriptions of the system, see [14,32,6,22]). Carroll assumes we have a finite number of candidates and a finite number of voters each having strict (no ties) preferences over the candidates. Carroll assigns to each candidate an integer that we will call the candidate's *Carroll score*. A *Condorcet winner* is a candidate who in pairwise elections with each candidate other than him- or herself receives strictly more than half the votes. The Carroll score of a candidate is the smallest number of sequential exchanges of adjacent candidates in voter preferences needed to make that candidate a Condorcet winner. Note that one exchange means the exchange of two adjacent candidates in the preference order of one voter. In the sample election given earlier, the Carroll scores are:

Candidate	Carroll score
Pat Buchanan	30,000,001
Al Gore	20,000,001
George W. Bush	10,000,001

Carroll's scheme declares the winner (or winners) to be whoever has the lowest Carroll score—that is, whoever is closest to being a Condorcet winner. So, in the given example, George W. Bush would be the winner under Carroll's election scheme.

Carroll's election scheme has many attractive properties from the political science point of view. Indeed, McLean and Urken include it in their collection of the gems of over two thousand years of social choice theory [31]. However, it is natural to ask whether the system is computationally tractable. Most crucially, how hard is it to test who won, and to test which of two candidates did better?

To study the complexity of these questions, one should formally describe each as a decision problem. Throughout, we assume that preference collections are coded as lists: The preferences of k voters will be coded as $\langle P_1, P_2, \dots, P_k \rangle$, where P_i is the permutation of the candidates reflecting the preferences of the i th voter.

Carroll Winner [6]

Instance: $\langle C, c, V \rangle$, where C is a candidate set, $c \in C$, and V is a preference collection over the candidate set C .

Question: Does candidate c win the Carroll election over candidate set C when the voters' preferences are V ? That is, does it hold that

$$(\forall d \in C)[\text{CarrollScore}(c) \leq \text{CarrollScore}(d)]?$$

Carroll Comparison [6]

Instance: $\langle C, c, d, V \rangle$, where C is a candidate set, c and d are candidates ($c, d \in C$), and V is a preference collection over the candidate set C .

Question: Does candidate d defeat c in the Carroll election over candidate set C when the voters' preferences are V ? That is, does it hold that

$$\text{CarrollScore}(d) < \text{CarrollScore}(c)?$$

The above two sets nicely capture the two most central questions one might ask about Carroll elections. But what is known about their complexity?

Before answering this, we quickly review some basic definitions and background from computational complexity theory.

Definition 1. 1. For any class C , we say that set A is C -hard iff $(\forall B \in C)[B \leq_m^p A]$.

2. For any class C , we say that set A is C -complete iff $A \in C$ and A is C -hard.

Definition 2. $P_{||}^{NP} = \bigcup_{A \in NP} \{L \mid L \leq_{tt}^p A\}$. (Recall that $\leq_{tt}^p$, polynomial-time truth-table reducibility [28], can be defined by $E \leq_{tt}^p F$ iff there is a polynomial-time machine M such that $L(M^F) = E$ and M asks all its questions to F in parallel and receives all their answers simultaneously.)

Parallel access to NP turns out to be identical to logarithmically bounded sequential access: $P_{||}^{NP} = P_{O(\log n)-T}^{NP}$ [19]. In fact, $P_{||}^{NP}$ has many characterizations (see [47]).

$P_{||}^{NP}$ plays a crucial role in complexity theory. For example, Kadin [25] has proven that if some sparse set is $\leq_T^p$ -complete for NP then $PH = P_{||}^{NP}$. Hemachandra and Wechsung [20] have shown that the theory of randomness (in the form of the resource-bounded Kolmogorov complexity theory of Adleman [1], Hartmanis [18], and Sipser [39]) is deeply tied to the question of whether $P_{||}^{NP} = P^{NP}$, i.e., whether parallel and sequential access to NP coincide. Buss and Hay [10] have shown that $P_{||}^{NP}$ exactly captures the class of sets acceptable via multiple rounds of parallel queries to NP and also exactly captures the disjunctive closure of the second level of the Boolean Hierarchy [11,12].

Notwithstanding all the above appearances of $P_{||}^{NP}$ in complexity theory, $P_{||}^{NP}$ was strangely devoid of natural complete problems. The class was known somewhat indirectly to have a variety of complete problems, but they were not overwhelmingly natural. In particular, a seminal paper by Wagner [46] proves that many questions regarding the parity of optimizations are complete for the class " P_{bf}^{NP} ," and $P_{bf}^{NP} = P_{||}^{NP}$ was proven soon thereafter (see the discussion of this in the footnote of [27]).

Happily, **Carroll Winner** and **Carroll Comparison** do provide complete problems for $P_{||}^{NP}$. And the naturalness of these complete problems is impossible to dispute, given that *the issues they capture were created about 100 years before NP or $P_{||}^{NP}$ were studied!*

Bartholdi, Tovey, and Trick [6] proved that **Carroll Winner** is NP-hard and that **Carroll Comparison** is coNP-hard.¹ They leave open the issue of whether either is complete for its class. Hemaspaandra, Hemaspaandra, and Rothe [22] resolved these questions, and provided natural complete problems for $P_{||}^{NP}$ as follows.²

Theorem 1 ([22]). *Carroll Winner and Carroll Comparison are $P_{||}^{NP}$ -complete.*

We conclude this section with some comments on and discussion of Theorem 1. First and most centrally, Theorem 1 shows that—though beautiful in political science terms—Carroll’s voting system is of distressingly high complexity in computational terms. Clearly, systems’ computational complexity should be weighed carefully when choosing electoral systems.

Second, one might ask why raising NP-hardness and coNP-hardness results to $P_{||}^{NP}$ -completeness results is valuable. There are two quite different explanations of why it is important to do this. One answer, which may be particularly attractive to theoretical computer scientists, is:

To understand a problem we seek to know not only how hard it is, but what the source/nature of its hardness is. The most central way that we classify the quintessential nature of a problem is to prove it complete for a class. In particular, when we prove a problem complete for $P_{||}^{NP}$, we know that what it is really about—the source of its hardness—is parallelized access to NP.

The second answer is a somewhat more practical one. $P_{||}^{NP}$ -completeness gives an upper and a lower bound, and each is useful. The upper bound limits the complexity of the problem. (In contrast, $\overline{\text{SAT}} \oplus \text{Halting Problem}$ is coNP-hard, but clearly it is much harder than **Carroll Comparison**, which is also coNP-hard.) And the $P_{||}^{NP}$ -hardness claim—the raised lower bound—may potentially be evidence that in certain alternative models of computing the problem may be harder than we might conclude from the weaker lower bound alone; this point is analyzed in detail by Hemaspaandra, Hemaspaandra, and Rothe [23].

Finally, regarding our $P_{||}^{NP}$ -completeness results, let us mention a worry. In our model, we looked at the preference sets of the voters and defined **Carroll Score** relative to that. But just how do we find these preferences? Of course, we could simply ask each voter. But what if the voters lie to us? In fact, do voters have an incentive to lie? If a voter knows the preferences of the other voters, how hard is it for him or her to compute what lie to tell about his or her preferences to get a desired outcome? These questions have been studied, for different systems, by Bartholdi, Tovey, and Trick [5].

¹ To be historically accurate, we mention that what they actually prove is an NP-hardness result for (essentially) the complement of **Carroll Comparison**. However, that is equivalent to the claim stated above.

² To be historically accurate, what they actually prove is $P_{||}^{NP}$ -completeness for (essentially) the complement of **Carroll Comparison**. However, the complement of a $P_{||}^{NP}$ -complete set is always $P_{||}^{NP}$ -complete.

3 An Optimal “Impracticality Theorem”

Arrow’s Theorem [2] states that no preference aggregation function has all of four natural properties (non-dictatorship, monotonicity, the Pareto Condition, and independence of irrelevant alternatives). This is often referred to as an “Impossibility Theorem.”

Bartholdi, Tovey, and Trick [6] stated and proved what they call an “Impracticality Theorem”—a theorem focusing on computational infeasibility.

Below, by election scheme we refer to preference aggregation schemes.

Definition 3 ([48]).

1. An election scheme is *neutral* if it is symmetric in the way it treats the candidates.
2. An election scheme is *Condorcet* if whenever there is a Condorcet winner that person is elected.
3. An election scheme is *consistent* if for every W_1 , W_2 , and Outcome such that
 - a) $W = W_1 \cup W_2$,
 - b) $W_1 \cap W_2 = \emptyset$,
 - c) the election scheme operating on preferences of voter set W_1 has the outcome Outcome, and
 - d) the election scheme operating on preferences of voter set W_2 has the outcome Outcome,
 then the election scheme operating on the preferences of voter set W also has the outcome Outcome.

Theorem 2 ([6] (Impracticality Theorem)). *For any election scheme that is neutral, Condorcet, and consistent, the winner problem (“Did candidate c win this election?”) is NP-hard.*

Bartholdi, Tovey, and Trick [6] pose as an open question whether stronger versions of this theorem can be established. In fact, one can optimally locate the degree of difficulty.

Theorem 3 ([21] (Optimal Impracticality Theorem)). *For any election scheme that is neutral, Condorcet, and consistent, the winner problem (“Did candidate c win this election?”) is $P_{||}^{\text{NP}}$ -complete.*

Theorems 2 and 3 show that achieving a reasonable degree of fairness (assuming neutrality, Condorcet-ness, and consistency are one’s notion of a reasonable degree of fairness) in polynomial time is impossible unless $P = NP$.

Corollary 1. *No neutral, Condorcet, consistent election scheme has a polynomial-time solvable winner problem unless $P = NP$.*

The above results are shown by the combination of a lot of work and a devilish sleight of hand. The sleight of hand is that, due to work of Young and Levenglick [48], it is known that there is a unique system that is neutral, Condorcet, and consistent. This system is known as Kemeny voting (see [26]). (Trivia fact: This is the same John Kemeny who developed the computer language “BASICS.”)

Kemeny elections work as follows. The outcome of an election is the collection of all (not necessarily strict) preference orders that are “closest” to the preference orders of the voters. Such a preference order is called a *Kemeny consensus*. Of course, there are different ways to define closeness. For Kemeny elections the goal is to minimize the sum of the distances to the preference order for each voter, where the distance between two preference orders P and P' is defined as follows: For every pair of candidates c and d , add 0 if P and P' have the same relative preference ($c < d$, $c = d$, $c > d$) on c and d , add 1 if c and d are tied in one of P and P' and not tied in the other, and add 2 if one of P and P' prefers c to d and the other prefers d to c .

The winner problem for Kemeny elections is thus:

Kemeny Winner [6]

Instance: $\langle C, c, V \rangle$, where C is a candidate set, $c \in C$, and V is a preference collection over the candidate set C .

Question: Does candidate c win the Kemeny election over candidate set C when the voters’ preferences are V ? That is, does there exist a Kemeny consensus such that c is the preferred candidate in the Kemeny consensus.

In a Kemeny election there may be more than one winner. Bartholdi, Tovey, and Trick [6] proved that **Kemeny Winner** is NP-hard. Hemaspaandra [21] strengthened this by proving that **Kemeny Winner** is $P_{||}^{NP}$ -complete. Theorem 3 follows.

4 Power and Apportionment

Let us consider the issue of proportional representation systems. That is, suppose in some country you have some political parties (Red, Green, etc.) and in an election each person votes for (exactly) one party. Given the results of the election (the vote total for each party) and the number of seats in the parliament, *how many seats should each party get?*

Let us now consider another problem, that of a federal system. That is, suppose that in some country you have some states, and under that country’s constitution every ten years a population count is done. Given the results of the census (the counted population of each state) and the number of seats in the parliament, *how many seats should each state get?*

Note that, except for terminology, these are the same problem. That is, if we view parties as states and party vote counts as state populations, the two

problems coincide.³ So we will henceforward use the latter terminology: “states” and “populations.” These problems are of great practical importance. For example, in the state+population context, this issue occurs in the United States every ten years in the apportionment of the House of Representatives of the US Congress. In fact, in the United States this apportionment of the House of Representatives not only is politically important in shaping the representation of each state in that House, but also determines the influence of each state in choosing the president. The reason for this latter effect is that Americans don’t vote directly for their president. Rather, each state is given a number of votes in the “Electoral College” equal to two plus the number of seats the state has in the House. The Electoral College then elects the president. In concept,⁴ a given Elector could, even if Al Gore did the best in the Elector’s state, validly vote for Darth Vader if Darth Vader met the Constitution’s qualification list:

No person except a natural born citizen, or a citizen of the United States at the time of the adoption of this Constitution, shall be eligible to the office of President; neither shall any person be eligible to that office who shall not have attained to the age of thirty five years, and been fourteen years a resident within the United States.

However, in practice, all a state’s Electors generally vote for the candidate who received the most votes in that state. (Even if Electors vote this way, in concept Al Gore could win the coming election, even if George W. Bush received substantially more votes.) This influence on the presidential selection process adds an extra importance to the already important issue of apportioning the House.

What constraints apply? To quote the US Constitution directly,

Representatives and direct taxes shall be apportioned among the several states which may be included within this union, according to their respective numbers, which shall be determined by adding to the whole number of free persons, including those bound to service for a term of years, and excluding Indians not taxed, three fifths of all other Persons.⁵ The actual Enumeration shall be made within three years after the first meeting of

³ By this, we mean that they are *mathematically* the same problem. It of course is possible that *politically* they may require different solutions [3, Chapters 11 and 12]. For example, in proportional representation systems many countries discourage fragmentation of parliament by setting a high lower bound (e.g., 5 percent) on what vote portion is needed to get any seats at all. However, similarly excluding small states from having votes in a federal system would be bizarre.

⁴ Some states try to restrict this via state law, but it is unlikely that those laws have force.

⁵ This sentence was modified by Section 2 of the Fourteenth Amendment: Representatives shall be apportioned among the several states according to their respective numbers, counting the whole number of persons in each state, excluding Indians not taxed. But when the right to vote at any election for the choice of electors for President and Vice President of the United States, Representatives in Congress, the executive and judicial officers of a state, or the members of the legislature thereof, is denied to any of the male inhabitants of such state, being twenty-one years of

the Congress of the United States, and within every subsequent term of ten years, in such manner as they shall by law direct. The number of Representatives shall not exceed one for every thirty thousand, but each state shall have at least one Representative; and until such enumeration shall be made, the state of New Hampshire shall be entitled to chuse [sic.] three, Massachusetts eight, Rhode Island and Providence Plantations one, Connecticut five, New York six, New Jersey four, Pennsylvania eight, Delaware one, Maryland six, Virginia ten, North Carolina five, South Carolina five, and Georgia three.

—United States Constitution, Article I, Section 2.

That is, the key constraints are: Each state’s number of representatives must be a nonnegative integer and (by current Federal law but not required by the Constitution—in fact, the number has changed over time) the total number of representatives must be 435. And there are additional technical requirements: Each state must be given at least one representative in the House and there may be at most one representative per thirty thousand people. However, these technical constraints almost never change the outcome, so let us in this paper ignore them (see [24] for a more detailed discussion) except where we explicitly mention them, namely, in one place later Wyoming would be given zero seats under one of the methods.

But what does it mean to “fairly” apportion the House? What does “fairness” mean? Suppose California’s population is exactly $51.4/435$ of the total US population, and thus California’s “quota” (its population times the House size divided by total population) is 51.4. It is natural, though as we’ll discuss later perhaps not right, to view “fair” as meaning that states get seat allocations that are close to the states’ quotas.

So, would it be fair for an apportionment method to assign California neither $\lceil 51.4 \rceil = 52$ nor $\lfloor 51.4 \rfloor = 51$ seats but rather to assign it 48 seats? Though one has to use artificial numbers to get this, the apportionment method currently used in the United States can do that. However, the reason it can do that actually makes sense. The method currently used is in some sense trying to avoid bad *ratios* of assigned seats to quota. California is so huge that, ratio-wise, even if it misses quota by a few seats that is not a disaster, especially if by doing so one can avoid more severe ratio problems elsewhere, e.g., if Wyoming deserves 1.8 and stealing a seat from California allows us to assign Wyoming 2 seats rather than 1 seat.

Of course, it is clear that we should not assign seats just by judging by eye. What we need is a formal, clear rule or algorithm. In fact, as mentioned in Section 1, many of the greatest figures in American history designed exactly such rules. Let us state these algorithms.

age [this is modified to eighteen years of age by the Twenty-Sixty Amendment], and citizens of the United States, or in any way abridged, except for participation in rebellion, or other crime, the basis of representation therein shall be reduced in the proportion which the number of such male citizens shall bear to the whole number of male citizens twenty-one years of age in such state.

Let H be the number of seats in the House. Let n be the number of states. Let p_i be the population of the i th state. Let

$$q_i = \frac{p_i \cdot H}{\sum_{1 \leq j \leq n} p_j}.$$

q_i is called the *quota* of state i . s_i will denote the number of seats given to state i .

Alexander Hamilton, in the late 1700s, proposed the following algorithm. Initially, set $s_i = \lfloor q_i \rfloor$. This assigns $\sum_{1 \leq j \leq n} \lfloor q_j \rfloor \leq H$ seats. We have left some nonnegative integer number of seats,

$$0 \leq H - \sum_{1 \leq j \leq n} \lfloor q_j \rfloor < n.$$

Set the remainder values $r_i = q_i - \lfloor q_i \rfloor$. Sort the remainder values of the n states from largest to smallest. For the $H - \sum_{1 \leq j \leq n} \lfloor q_j \rfloor$ states with the largest values of r_i , set $s_i = s_i + 1$. This completes the algorithm. In other words, Hamilton gives each state the floor of its quota and then parcels out the remaining seats to the states with the biggest fractional parts of their quotas.⁶ Hamilton's algorithm is clear and simple to implement. It has the lovely property of "obeying quota": Each state is given either the floor or the ceiling of its quota.

Five other methods—those of Adams (the sixth president), Dean (a professor at the University of Virginia and Dartmouth in the nineteenth century), Huntington-Hill (Hill was Chair of the House Committee on the Census, and Huntington was a Harvard professor, both circa the first half of the twentieth century), Webster (the great lexicographer, and also a senator), and Jefferson (the third president)—all may be viewed as sharing the same framework as each other, and differ only in which of five methods are used to round numbers to integer values, namely, respectively, taking the ceiling, taking the harmonic mean, taking the geometric mean, taking the arithmetic mean, and taking the floor.

Let us be a bit more specific (but for a detailed specification of these algorithms see Balinski and Young [3]). Consider the following five ways of cutting up the nonnegative reals into segments.

Adams:

$$\begin{aligned} S_0 &= \{0\}. \\ S_i &= (i - 1, i], \text{ for } i \geq 1. \end{aligned}$$

Dean:

$$\begin{aligned} S_0 &= \emptyset. \\ S_i &= \left[\frac{(i-1)i}{i-\frac{1}{2}}, \frac{i(i+1)}{i+\frac{1}{2}} \right), \text{ for } i \geq 1. \end{aligned}$$

Huntington-Hill:

$$\begin{aligned} S_0 &= \emptyset. \\ S_i &= \left[\sqrt{(i-1)i}, \sqrt{i(i+1)} \right), \text{ for } i \geq 1. \end{aligned}$$

⁶ This isn't quite a fully specified algorithm as if states have identical fractional parts this does not state which way to sort them relative to each other. However, such ties are extremely unlikely in a large-population country, so for simplicity throughout this exposition we assume that ties do not occur.

Webster:

$$S_0 = [0, \frac{1}{2}).$$

$$S_i = [i - \frac{1}{2}, i + \frac{1}{2}), \text{ for } i \geq 1.$$

Jefferson:

$$S_i = [i, i + 1), \text{ for } i \geq 0.$$

Each of these five ways of cutting up the line gives an algorithm. Namely, via binary search, find⁷ a real number d such that

$$H = \sum_{0 \leq j \leq n} h(\frac{p_j}{d}),$$

where, for any nonnegative real r , $h(r)$ denotes the unique integer such that $r \in S_{h(r)}$.

These five methods are known⁸ as sliding-divisor methods. One slides around the divisor d , each time rounding $\frac{p_i}{d}$ up or down based on $h(\cdot)$, i.e., based on ceiling, floor, or one of the three types of means. Note that the rounding method can have a major impact on how the method gives and withholds votes. For example, Jefferson, who was from the very populous state of Virginia, takes floors. So a small state won't get its second seat until $\frac{p_i}{d} \geq 2$. Of the five methods, Jefferson's is the harshest to small states and Adams is the most generous to small states, and the opposite holds towards large states. For example, under the 1990 census, in which California's quota was 52.185, Jefferson gives California 54 seats, Webster, Hamilton, Hill-Huntington, and Dean each give California 52 seats, and Adams gives California 50 seats.

Though space does not permit its inclusion in this article, the historical story of the discussions on which method to use is simply amazing. Great historical figures built rules/algorithms. Great historical figures (e.g., George Washington!) judged and rejected rules/algorithms. And, human nature being what it is, perhaps it isn't too shocking that at times in the discussions people happened to argue in favor of apportionment algorithms that happened to shift a seat or two in a direction corresponding to their interests. The book "Fair Representation:

⁷ It is possible that if state populations share common divisors in unlucky ways no such d will exist, but let us for simplicity assume that does not happen.

⁸ Some of the methods were discovered independently by others, and thus are known under various names (see the discussion and history in [3]). Jefferson's method is also known as d'Hondt's method, after Victor d'Hondt, who rediscovered it in the late 1800s, and is also known as the method of greatest divisors and the method of highest averages. Webster's method was also discovered by Jean-André Sainte-Laguë and is known also by that name and as the method of odd numbers. Hamilton's method is also known, naturally enough, as the method of largest remainders and the method of greatest remainders.

Also, throughout, we use the notions X 's algorithm and X 's method interchangeably. However, to be quite accurate, we should mention that in some cases methods were specified or described in different terms that happen to define the same outcome as that of the algorithms specified, rather than being directly defined via the above algorithms.

Meeting the Ideal of One Man, One Vote” ([3], see also [4]) tells this fascinating history in a delightful, gripping fashion and also excellently presents the methods and the mathematics involved. Balinski and Young [3] is required (and charming) reading for anyone interested in this problem.

However, let us step back and look at the big picture: “fairness.” What is fairness? In fact, Hamilton’s method and the five sliding divisor methods are all driven by different versions of the same intuitive feeling about what fairness is. They all have the flavor of viewing fairness as meaning that a state’s seat allocation should be close to its quota. All they differ on is their notion of closeness. For example, Hamilton keeps the absolute discrepancy below one but may allow larger relative discrepancies; in contrast, each of the five sliding-divisor methods allows absolute discrepancies above one, but each has a theoretical interpretation as minimizing inequality with respect to some notion (different in each of the five cases) of inequality driven by relative discrepancy [4, Section 6].⁹

But is closeness of seats to quota really the right notion of fairness? Let us for a moment switch from state+population terminology to party+vote terminology as it makes these examples clearer.¹⁰ Suppose our country has three parties, Green, Red, and Blue, and a 15-seat majority-vote-wins parliament in which (due to excellent party discipline) all members of a party always vote the same way (e.g., as the party leadership dictates). Suppose the seat allocations are:

Green 8 seats
Red 5 seats
Blue 2 seats,

and that these seat allocations are good matches to the parties’ quotas. So, between them, the Red and Blue parties received almost half the vote. Yet they have no power—none at all! The Green party is a self-contained majority and thus controls every issue.

⁹ Of course, nearness to quota was not the sole metric that was used to judge methods. Many authors have been strongly influenced by certain undesirable behaviors some methods display. For example, under the 1880 census, apportioning a 299 seat House under Hamilton’s method gives Alabama 8 seats. But Hamilton’s method applied to a 300-seat House gives Alabama 7 seats! This became known as the *Alabama Paradox*.

This is of course counterintuitive but is not a real paradox. Increasing the house size increases the quota of every state equally in relative terms, so it increases the quotas of the larger states more than smaller states in absolute terms, so it is quite possible for (with no changes in the integer parts) a larger state’s fractional quota part ($q_i - [q_i]$) to cross from being less than Alabama’s to being greater than Alabama’s when the House size goes from 299 to 300.

¹⁰ Even for states the analytical approach we will take is a reasonable way of asking what the power of a given state is on issues where interests are heavily state-inspired (e.g., formulas to distribute federal dollars among the states), and in the model in which all possible state-to-issue preferences are viewed as equally likely.

Now suppose that the seat allocations are:

Green 7 seats
Red 5 seats
Blue 3 seats,

and that these seat allocations are good matches to the parties' quotas. (Note that this can happen due to, relative to the previous example, a small popularity shift from Green to Blue.) Green now has more than twice the seats of Blue. Yet their power is exactly equal! To get a majority one needs two parties to agree; no one party is a majority, and *any* two parties are a majority. So all three parties have identical power.

What these two examples show is that (portion of) seats is a dreadful predictor of (portion of) power. Yet all the apportionment methods focus on matching quotas to seats, rather than matching quotas to power. This latter seems (at least in a general setting, though perhaps less so for the US House, depending on one's feeling for what "according to their respective numbers" means) very natural and very desirable. Can it be done? Are there computational-complexity impediments? And what does "power" really mean anyway?

Let us start with that last question. What the Green/Red/Blue examples make clear is that power is not about number of seats, but rather has something to do with the pattern of seat distribution: the way blocks of parties, but let us now switch back to the state+population terminology and say blocks of states, can form majorities. This can be formalized crisply and mathematically, and that was done starting in the twentieth century, via the theory of power indices (see [37,15,36]).

For example, consider the following notion. We will say a state is *critical* if (in a given vote) it is on the winning side but if it changes its vote that side no longer wins. The (unnormalized) Banzhaf power index is the probability that in a random vote the state is critical. Viewed in a different scaling, it is the number of the 2^n ways the n states can vote yes/no in which the given state is essential to the win.

There are a large number of other power indices, each trying to capture some natural view of power. For example, in the Banzhaf index, we view each state as voting yes or no, with 2^n equally likely overall possibilities, and critical states as our focus. In the model underlying the other most widely discussed (and different [40]) index, the Shapley-Shubik power index [37], we view each state as having some degree of preference for a given bill, and we view each of the $n!$ different preference sequences (no ties) as equally likely, and we count in how many of the $n!$ orders a given state is a *pivot*, that is, is the unique state such that the votes of all states whose preference strength is greater than the pivot's do not make up a majority, but when the pivot's votes are added become a majority. (The pivot state is the swing vote that the states with more extreme preferences will court.)

So, there is a large, lovely, already developed theory of power indices. We need only allocate in such a way as to closely match quotas to (rescaled) power—that is, to seek to have each state’s power correspond to its portion of the population—and we are done. But is this an easy task? Unfortunately, the task may not be so easy. Let us consider the Valiant’s counting class $\#P$, which counts the accepting paths of NP machines.

Definition 4 ([44,45]). *A function $f : \Sigma^* \rightarrow \mathbf{N}$ is in $\#P$ if there is a non-deterministic polynomial-time Turing machine N such that*

$$(\forall x)[f(x) = \#acc_N(x)],$$

where $\#acc_N(x)$ denotes the number of accepting paths of $N(x)$.

$\#P$ is a tremendously powerful class. Toda ([42], see also [8,43,35]) showed that Turing access to $\#P$ suffices to accept every set in the polynomial hierarchy.

Unfortunately, it is well-known that computing power indices is typically $\#P$ -complete. Prasad and Kelly [34] proved (surprisingly recently) that the Banzhaf index is $\#P$ -complete, and it has long been known that the Shapley-Shubik power index is $\#P$ -complete [17]. And even if this were not catastrophic in and of itself, and if we by magic did have a polynomial-time algorithm to compute these power indices (which would require $P = P^{\#P}$ and thus certainly $P = NP$), we would still be faced with a horridly broad search over the set of all possible apportionments whose power-fairness to evaluate (though of course $P = NP$ would also slam-dunk this).

The “magic” path mentioned above is unlikely to exist. Nonetheless, the above tasks have been tackled, albeit via a combination of heuristics, algorithmics, and experimentation. In particular, Hemaspaandra, Rajasethupathy, Sethupathy, and Zimand [24] have combined a dynamic-programming computation of power indices (see also [30], cf. [29]) with simulated annealing search over seat apportionments driven by the goal of matching rescaled power with quota, all compared under the standard difference norms. They do this comparison for every single one of the twenty-one actual censuses—1790, 1800, 1890, . . . , 1970, 1980, and 1990¹¹—against each of the six classic apportionment methods.

What they find is that the heuristic approach in every case—every census, every metric—yields provably fairer results than the other methods. For example, they construct an apportionment A' of the House under the 1990 census data and exactly compute the power indices of each state, and that apportionment A' is such that the correspondence between (rescaled) power and quota is (much) higher than the same correspondence for each of the other methods as calculated via the exactly computed power indices of each state under these methods.

It is important to keep in mind that Hemaspaandra et al. [24] is an experimental algorithms paper. They in no way claim that A' is an *optimal* apportionment. What they show is merely that in a rigorous, mathematically well-defined sense it is superior to the apportionments given by the other six methods.

¹¹ The 2000 census figures are not yet available.

Table 1. The apportionments for the 1990 census, using the six classic algorithms and using the heuristic method “Banzhaf”

State	Population	Banzhaf	Adams	Dean	Hunt.-Hill	Webster	Jefferson	Hamilton
CA	29760021	47	50	52	52	52	54	52
NY	17990455	31	30	31	31	31	33	32
TX	16986510	29	29	30	30	30	31	30
FL	12937927	22	22	23	23	23	23	23
PA	11881643	21	20	21	21	21	21	21
IL	11430602	20	19	20	20	20	21	20
OH	10847115	19	18	19	19	19	19	19
MI	9295297	16	16	16	16	16	17	16
NJ	7730188	14	13	13	13	14	14	14
NC	6628637	12	11	12	12	12	12	12
GA	6478216	12	11	11	11	11	11	11
VA	6187358	11	11	11	11	11	11	11
MA	6016425	11	10	10	11	11	11	11
IN	5544159	10	10	10	10	10	10	10
MO	5117073	9	9	9	9	9	9	9
WI	4891769	9	9	9	9	9	9	9
TN	4877185	9	9	9	9	9	8	9
WA	4866692	9	9	8	8	9	8	8
MD	4781468	9	8	8	8	8	8	8
MN	4375099	8	8	8	8	8	8	8
LA	4219973	8	8	7	7	7	7	7
AL	4040587	7	7	7	7	7	7	7
KY	3685296	7	7	6	6	6	6	6
AZ	3665228	7	7	6	6	6	6	6
SC	3486703	6	6	6	6	6	6	6
CO	3294394	6	6	6	6	6	6	6
CT	3287116	6	6	6	6	6	6	6
OK	3145585	6	6	6	6	5	5	5
OR	2842321	5	5	5	5	5	5	5
IA	2776755	5	5	5	5	5	5	5
MS	2573216	5	5	5	5	4	4	4
KS	2447574	4	5	4	4	4	4	4
AR	2350725	4	4	4	4	4	4	4
WV	1793477	3	3	3	3	3	3	3
UT	1722850	3	3	3	3	3	3	3
NE	1578385	3	3	3	3	3	2	3
NM	1515069	3	3	3	3	3	2	3
ME	1227928	2	3	2	2	2	2	2
NV	1201833	2	2	2	2	2	2	2
NH	1109252	2	2	2	2	2	2	2
HI	1108229	2	2	2	2	2	2	2
ID	1006749	2	2	2	2	2	1	2
RI	1003464	2	2	2	2	2	1	2
MT	799065	1	2	2	1	1	1	1
SD	696004	1	2	1	1	1	1	1
DE	666168	1	2	1	1	1	1	1
ND	638800	1	2	1	1	1	1	1
VT	562758	1	1	1	1	1	1	1
AK	550043	1	1	1	1	1	1	1
WY	453588	1	1	1	1	1	0	1
Totals	248072974	435	435	435	435	435	435	435

Table 2. The (rescaled) power, under the Banzhaf power index with respect to the L2 norm, for the apportionments for the 1990 census under the different methods

State	Quota	Banzhaf	Adams	Dean	Hunt.-Hill	Webster	Jefferson	Hamilton
CA	52.1847	51.9091	56.3941	58.8488	58.8256	58.7932	61.2532	58.6931
NY	31.5466	31.4477	30.1897	31.1393	31.1401	31.1409	33.1562	32.2155
TX	29.7861	29.2732	29.1229	30.0761	30.0767	30.0773	31.0326	30.0818
FL	22.6869	21.8830	21.8016	22.7648	22.7655	22.7660	22.7033	22.7660
PA	20.8347	20.8528	19.7591	20.7233	20.7239	20.7244	20.6716	20.7244
IL	20.0437	19.8279	18.7446	19.7091	19.7097	19.7102	20.6716	19.7102
OH	19.0206	18.8079	17.7344	18.6991	18.6996	18.7002	18.6559	18.7001
MI	16.2995	15.7744	15.7255	15.6914	15.6919	15.6924	16.6546	15.6924
NJ	13.5550	13.7713	12.7376	12.7127	12.7131	13.7037	13.6761	13.7037
NC	11.6234	11.7810	10.7599	11.7251	11.7256	11.7260	11.7037	11.7260
GA	11.3597	11.7810	10.7599	10.7400	10.7404	10.7408	10.7208	10.7408
VA	10.8496	10.7900	10.7599	10.7400	10.7404	10.7408	10.7208	10.7408
MA	10.5499	10.7900	9.7745	9.7569	10.7404	10.7408	10.7208	10.7408
IN	9.7218	9.8015	9.7745	9.7569	9.7573	9.7577	9.7399	9.7577
MO	8.9729	8.8152	8.7913	8.7758	8.7761	8.7765	8.7609	8.7765
WI	8.5778	8.8152	8.7913	8.7758	8.7761	8.7765	8.7609	8.7765
TN	8.5522	8.8152	8.7913	8.7758	8.7761	8.7765	7.7834	8.7765
WA	8.5338	8.8152	8.7913	7.7964	7.7967	8.7765	7.7834	7.7971
MD	8.3844	8.8152	7.8098	7.7964	7.7967	7.7970	7.7834	7.7971
MN	7.6718	7.8308	7.8098	7.7964	7.7967	7.7970	7.7834	7.7971
LA	7.3998	7.8308	7.8098	6.8185	6.8188	6.8191	6.8074	6.8191
AL	7.0852	6.8482	6.8301	6.8185	6.8188	6.8191	6.8074	6.8191
KY	6.4622	6.8482	6.8301	5.8420	5.8422	5.8425	5.8326	5.8425
AZ	6.4270	6.8482	6.8301	5.8420	5.8422	5.8425	5.8326	5.8425
SC	6.1140	5.8671	5.8517	5.8420	5.8422	5.8425	5.8326	5.8425
CO	5.7768	5.8671	5.8517	5.8420	5.8422	5.8425	5.8326	5.8425
CT	5.7640	5.8671	5.8517	5.8420	5.8422	5.8425	5.8326	5.8425
OK	5.5158	5.8671	5.8517	5.8420	5.8422	4.8670	4.8589	4.8670
OR	4.9841	4.8873	4.8746	4.8666	4.8668	4.8670	4.8589	4.8670
IA	4.8691	4.8873	4.8746	4.8666	4.8668	4.8670	4.8589	4.8670
MS	4.5122	4.8873	4.8746	4.8666	4.8668	3.8925	3.8861	3.8925
KS	4.2919	3.9085	4.8746	3.8922	3.8923	3.8925	3.8861	3.8925
AR	4.1220	3.9085	3.8984	3.8922	3.8923	3.8925	3.8861	3.8925
WV	3.1449	2.9307	2.9231	2.9185	2.9186	2.9187	2.9139	2.9187
UT	3.0210	2.9307	2.9231	2.9185	2.9186	2.9187	2.9139	2.9187
NE	2.7677	2.9307	2.9231	2.9185	2.9186	2.9187	1.9423	2.9187
NM	2.6567	2.9307	2.9231	2.9185	2.9186	2.9187	1.9423	2.9187
ME	2.1532	1.9534	2.9231	1.9453	1.9454	1.9455	1.9423	1.9455
NV	2.1074	1.9534	1.9484	1.9453	1.9454	1.9455	1.9423	1.9455
NH	1.9451	1.9534	1.9484	1.9453	1.9454	1.9455	1.9423	1.9455
HI	1.9433	1.9534	1.9484	1.9453	1.9454	1.9455	1.9423	1.9455
ID	1.7654	1.9534	1.9484	1.9453	1.9454	1.9455	0.9711	1.9455
RI	1.7596	1.9534	1.9484	1.9453	1.9454	1.9455	0.9711	1.9455
MT	1.4012	0.9766	1.9484	1.9453	0.9726	0.9727	0.9711	0.9727
SD	1.2205	0.9766	1.9484	0.9726	0.9726	0.9727	0.9711	0.9727
DE	1.1681	0.9766	1.9484	0.9726	0.9726	0.9727	0.9711	0.9727
ND	1.1201	0.9766	1.9484	0.9726	0.9726	0.9727	0.9711	0.9727
VT	0.9868	0.9766	0.9741	0.9726	0.9726	0.9727	0.9711	0.9727
AK	0.9645	0.9766	0.9741	0.9726	0.9726	0.9727	0.9711	0.9727
WY	0.7954	0.9766	0.9741	0.9726	0.9726	0.9727	0.0000	0.9727
Totals	435.0000	435.0000	435.0000	435.0000	435.0000	435.0000	435.0000	435.0000

Table 1 shows the apportionments for the 1990 census, using the six classic algorithms, and using the heuristic method (“Banzhaf”) based on the Banzhaf power index with respect to the L2 norm. Though the Constitution doesn’t allow zero seats to be given, the Hemaspaandra et al. study removes that constraint so as not to let that artificially bias against methods that might naturally assign that—in this case, Wyoming under the Jefferson algorithm. Table 2 shows the quotas and the rescaled powers.

The results are rather surprising. Among the classic algorithms, Adams’s algorithm is by far the harshest towards large states. (The ceiling in Adams’s algorithm lets the smallest states get, for example, their second vote quite easily.) Yet, it is clear from Table 2 that even Adams’s algorithm, which most authors view as horribly biased against the largest states, in fact is horribly biased *towards* the largest state—and the other five classic algorithms are even more horribly biased towards the largest state.

Here is a very informal way of understanding this. Consider a country with one huge state, A , with lots of seats in parliament, and m states each with one seat in parliament. So the total number of seats is $m + s_A$. In a random vote, what is the expected value of

| number among the m states voting yes – number of the m states voting no |?

This is just the absolute value of the expected distance that an m -step random walk ends up from the origin. So the answer to our expected vote gap question is $\Theta(\sqrt{m})$ (see [16]). Thus the small states will cancel each other out wildly—leaving big state A disproportionately likely to control the outcome since its s_A votes are cast as a monolithic block. For example, if $s_A = \sqrt{m} \log m$, then thinking a bit about the worst possible way the expectation could be realized (namely, 0 for β of the time and $\sqrt{m} \log m$ for $1 - \beta$ of the time and solving for β) it holds that state A controls the election with probability $1 - \mathcal{O}(\frac{1}{\log m})$ even though state A has only a tiny portion,

$$\frac{1}{1 + \sqrt{m}/\log m},$$

of the total vote.

This example gives some slight intuition as to why the power-based methods give fewer seats to a huge state, though on the other hand, as the second Green/Red/Blue example shows, in some setting being bigger yields relatively too little power rather than too much. Power indices are subtle objects that, as their #P-completeness makes clear (unless $P = P^{\#P}$), are to some degree computationally opaque. Nonetheless, as the heuristic method of Hemaspaandra et al. is doing exact power computations, it directly responds to the bumps and curves that power indices throw its way. And by doing so, the method constructs an apportionment of the House that is provably fairer than the one currently in use.

5 Conclusion

Theoretical computer science and the study of elections have much to offer to each other. Theoretical computer science, which has a strong history of studying diverse domains ranging from cryptography to quantum computing, can much benefit from the challenges and opportunities posed by such a long-established, natural, important, and often beautifully formalized domain as political science, and in particular the theory of elections. For example, as discussed in Section 2, the theory of elections provided the first natural problem complete for parallelized NP. Looking in the opposite direction, the theory of elections can benefit from the insights into electoral systems' feasibility—which should be a factor in designing and evaluating election systems—offered by the tools and techniques of theoretical computer science.

Acknowledgments. We thank our colleagues on the research projects discussed here: P. Rajasethupathy, J. Rothe, K. Sethupathy, and M. Zimand. The second author thanks M. Balinski for, twenty years ago, introducing him to both research and the study of elections.

References

1. L. Adleman. Time, space, and randomness. Technical Report MIT/LCS/TM-131, MIT, Cambridge, MA, April 1979.
2. K. Arrow. *Social Choice and Individual Values*. John Wiley and Sons, 1951 (revised edition, 1963).
3. M. Balinski and H. Young. *Fair Representation: Meeting the Ideal of One Man, One Vote*. Yale University Press, New Haven, 1982.
4. M. Balinski and H. Young. Fair representation: Meeting the ideal of one man, one vote. In H. Young, editor, *Fair Allocation*, pages 1–29. American Mathematical Society, 1985. Proceedings of Symposia in Applied Mathematics, V. 33.
5. J. Bartholdi, III, C. Tovey, and M. Trick. The computational difficulty of manipulating an election. *Social Choice and Welfare*, 6:227–241, 1989.
6. J. Bartholdi III, C. Tovey, and M. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6:157–165, 1989.
7. J. Bartholdi III, C. Tovey, and M. Trick. How hard is it to control an election? *Mathematical and Computer Modeling*, 16(8/9):27–40, 1992.
8. R. Beigel, L. Hemachandra, and G. Wechsung. Probabilistic polynomial time is closed under parity reductions. *Information Processing Letters*, 37(2):91–94, 1991.
9. D. Black. *Theory of Committees and Elections*. Cambridge University Press, 1958.
10. S. Buss and L. Hay. On truth-table reducibility to SAT. *Information and Computation*, 91(1):86–102, 1991.
11. J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.
12. J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy II: Applications. *SIAM Journal on Computing*, 18(1):95–111, 1989.

13. M. J. A. N. de Caritat, Marquis de Condorcet. *Essai sur l'Application de L'Analyse à la Probabilité des Décisions Rendues à la Pluraliste des Voix*. 1785. Facsimile reprint of original published in Paris, 1972, by the Imprimerie Royale.
14. C. Dodgson. A method of taking votes on more than two issues, 1876. Pamphlet printed by the Clarendon Press, Oxford, and headed “not yet published” (see the discussions in [31,9], both of which reprint this paper).
15. P. Dubey and L. Shapley. Mathematical properties of the Banzhaf power index. *Mathematics of Operations Research*, 4(2):99–131, May 1979.
16. W. Feller. *An introduction to probability theory and its applications*. Wiley, New York, 1968.
17. M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
18. J. Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *Proceedings of the 24th IEEE Symposium on Foundations of Computer Science*, pages 439–445. IEEE Computer Society Press, 1983.
19. L. Hemaspaandra. The strong exponential hierarchy collapses. *Journal of Computer and System Sciences*, 39(3):299–322, 1989.
20. L. Hemaspaandra and G. Wechsung. Kolmogorov characterizations of complexity classes. *Theoretical Computer Science*, 83:313–322, 1991.
21. E. Hemaspaandra. The complexity of Kemeny elections. In preparation.
22. E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Exact analysis of Dodgson elections: Lewis Carroll’s 1876 voting system is complete for parallel access to NP. *Journal of the ACM*, 44(6):806–825, 1997.
23. E. Hemaspaandra, L. Hemaspaandra, and J. Rothe. Raising NP lower bounds to parallel NP lower bounds. *SIGACT News*, 28(2):2–13, 1997.
24. L. Hemaspaandra, K. Rajasehupathy, P. Sethupathy, and M. Zimand. Power balance and apportionment algorithms for the United States Congress. *ACM Journal of Experimental Algorithmics*, 3(1), 1998. URL <http://www.jea.acm.org/1998/HemaspaandraPower>, 16pp.
25. J. Kadin. $P^{NP[\log n]}$ and sparse Turing-complete sets for NP. *Journal of Computer and System Sciences*, 39(3):282–298, 1989.
26. J. Kemeny and L. Snell. *Mathematical Models in the Social Sciences*. Ginn, 1960.
27. J. Köbler, U. Schöning, and K. Wagner. The difference and truth-table hierarchies for NP. *RAIRO Theoretical Informatics and Applications*, 21:419–435, 1987.
28. R. Ladner, N. Lynch, and A. Selman. A comparison of polynomial time reducibilities. *Theoretical Computer Science*, 1(2):103–124, 1975.
29. I. Mann and L. Shapley. Values of large games, IV: Evaluating the electoral college by Monte Carlo techniques. Research Memorandum RM-2651 (ASTIA No. AD 246277), The Rand Corporation, Santa Monica, CA, September 1960.
30. I. Mann and L. Shapley. Values of large games, VI: Evaluating the electoral college exactly. Research Memorandum RM-3158-PR, The Rand Corporation, Santa Monica, CA, 1962.
31. I. McLean and A. Urken. *Classics of Social Choice*. University of Michigan Press, 1995.
32. R. Niemi and W. Riker. The choice of voting systems. *Scientific American*, 234:21–27, 1976.
33. C. Papadimitriou and M. Yannakakis. On complexity as bounded rationality. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 726–733. ACM Press, 1994.
34. K. Prasad and J. Kelly. NP-completeness of some problems concerning voting games. *International Journal of Game Theory*, 19:1–9, 1990.

35. K. Regan and J. Royer. On closure properties of bounded two-sided error complexity classes. *Mathematical Systems Theory*, 28(3):229–244, 1995.
36. L. Shapley. Measurement of power in political systems. In W. Lucas, editor, *Game Theory and its Applications*, pages 69–81. American Mathematical Society, 1981. Proceedings of Symposia in Applied Mathematics, V. 24.
37. L. Shapley and M. Shubik. A method of evaluating the distribution of power in a committee system. *American Political Science Review*, 48:787–792, 1954.
38. H. Simon. *The Sciences of the Artificial*. MIT Press, 1969. Second edition, 1981.
39. M. Sipser. Borel sets and circuit complexity. In *Proceedings of the 15th ACM Symposium on Theory of Computing*, pages 61–69. ACM Press, 1983.
40. P. Straffin, Jr. Homogeneity, independence, and power indices. *Public Choice*, 30 (Summer), 1977.
41. United States Department of Commerce et al. versus Montana et al. US Supreme Court Case 91-860. Decided March 31, 1992.
42. S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 20(5):865–877, 1991.
43. S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM Journal on Computing*, 21(2):316–328, 1992.
44. L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.
45. L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.
46. K. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51(1–2):53–80, 1987.
47. K. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.
48. H. Young and A. Levenglick. A consistent extension of Condorcet’s election principle. *SIAM Journal on Applied Mathematics*, 35(2):285–300, 1978.

0-1 Laws for Fragments of Existential Second-Order Logic: A Survey

Phokion G. Kolaitis^{*1} and Moshe Y. Vardi^{**2}

¹ University of California, Santa Cruz kolaitis@cse.ucsc.edu

² Rice University vardi@cs.rice.edu

Abstract. The probability of a property on the collection of all finite relational structures is the limit as $n \rightarrow \infty$ of the fraction of structures with n elements satisfying the property, provided the limit exists. It is known that the 0-1 law holds for every property expressible in first-order logic, i.e., the probability of every such property exists and is either 0 or 1. Moreover, the associated decision problem for the probabilities is solvable.

In this survey, we consider fragments of existential second-order logic in which we restrict the patterns of first-order quantifiers. We focus on fragments in which the first-order part belongs to a prefix class. We show that the classifications of prefix classes of first-order logic with equality according to the solvability of the finite satisfiability problem and according to the 0-1 law for the corresponding Σ_1^1 fragments are identical, but the classifications are different without equality.

1 Introduction

In recent years a considerable amount of research activity has been devoted to the study of the model theory of finite structures [EF95]. This theory has interesting applications to several other areas including database theory [AHV95] and complexity theory [Imm98]. One particular direction of research has focused on the asymptotic probabilities of properties expressible in different languages and the associated decision problem for the values of the probabilities [Com88].

In general, if C is a class of finite structures over some vocabulary and if P is a property of some structures in C , then the *asymptotic probability* $\mu(P)$ on C is the limit as $n \rightarrow \infty$ of the fraction of the structures in C with n elements which satisfy P , provided that the limit exists. We say that P is *almost surely true* on C in case $\mu(P)$ is equal to 1. Combinatorialists have studied extensively the asymptotic probabilities of interesting properties on the class G of all finite graphs. It is, for example, well known and easy to prove that $\mu(\text{connectivity})=1$, while $\mu(k\text{-colorability})=0$, for every $k > 0$ [Bol85]. A theorem of Pósa [Pos76] implies that $\mu(\text{Hamiltonicity})=1$.

Glebskii et al. [GKLT69] and independently Fagin [Fag76] were the first to establish a fascinating connection between logical definability and asymptotic probabilities. More specifically, they showed that if C is the class of all finite structures over some relational

* Work partially supported by NSF grants CCR-9610257 and CCR-9732041.

** Work partially supported by NSF grant CCR-9700061. Work partly done at LIFO, University of Orléans

vocabulary and if P is an arbitrary property expressible in first-order logic (with equality), then $\mu(P)$ exists and is either 0 or 1. This result is known as *the 0-1 law for first-order logic*. The proof of the 0-1 law also implies that the decision problem for the value of the probabilities of first-order sentences is solvable. This should be contrasted with Trakhtenbrot's [Tra50] classical theorem to the effect that the set of first-order sentences which are true on *all* finite relational structures is unsolvable, assuming that the vocabulary contains at least one binary relation symbol.

It is well known that first-order logic has very limited expressive power on finite structures (cf. [EF95]). For this reason, one may want to investigate asymptotic probabilities for higher-order logics. Unfortunately, it is easy to see that the 0-1 law fails for second-order logic; for example, parity is definable by an existential second-order sentence. Moreover, the 0-1 law fails even for existential monadic second-order logic [KS85, Kau87]. In view of this result, it is natural to ask: are there fragments of second order-logic for which a 0-1 law holds?

The simplest and most natural fragments of second-order logic are formed by considering second-order sentences with only existential second-order quantifiers or with only universal second-order quantifiers. These are the well known classes of Σ_1^1 and Π_1^1 sentences respectively. Fagin [Fag74] proved that a property is Σ_1^1 definable if and only if it is NP-computable. As we observed, the 0-1 law fails for Σ_1^1 in general (and consequently for Π_1^1 as well). Moreover, it is not hard to show that the Σ_1^1 sentences having probability 1 form an unsolvable set.

In view of these facts, we concentrate on fragments of Σ_1^1 sentences in which we restrict the pattern of the first-order quantifiers that occur in the sentence. If $\mathcal{F}$ is a class of first-order sentences, then we denote by $\Sigma_1^1(\mathcal{F})$ the class of all Σ_1^1 sentences whose first-order part is in $\mathcal{F}$. Two remarks are in order now. First, if $\mathcal{F}$ is the class of all $\exists^*\forall^*\exists^*$ first-order sentences (that is to say, first-order sentences whose quantifier prefix consists of a string of existential quantifiers, followed by a string of universal quantifiers, followed by a string of existential quantifiers), then $\Sigma_1^1(\mathcal{F})$ has the same expressive power as the full Σ_1^1 . In other words, every Σ_1^1 formula is equivalent to one of the form $\exists\mathbf{S}\exists\mathbf{x}\forall\mathbf{y}\exists\mathbf{z}\theta(\mathbf{S}, \mathbf{x}, \mathbf{y}, \mathbf{z})$, where θ is a quantifier-free formula, $\mathbf{S}$ is a sequence of second-order relation variables and $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are sequences of first-order variables (*Skolem normal form*). Second, if $\phi(\mathbf{S})$ is a first-order sentence without equality over the vocabulary $\mathbf{S}$, then $\mu(\exists\mathbf{S}\phi(\mathbf{S})) = 1$ if and only if $\phi(\mathbf{S})$ is finitely satisfiable. Thus, for every first-order class $\mathcal{F}$, the decision problem for $\Sigma_1^1(\mathcal{F})$ sentences having probability 1 is at least as hard as the finite satisfiability problem for sentences in $\mathcal{F}$. The latter problem is known to be unsolvable [Tra50], even in the case where $\mathcal{F}$ is the class of $\exists^*\forall^*\exists^*$ sentences ([BGG97]). As a result, in order to pursue positive solvability results one has to consider fragments $\Sigma_1^1(\mathcal{F})$, where $\mathcal{F}$ is a class for which the finite satisfiability problem is solvable. Such classes $\mathcal{F}$ of first-order sentences are said to be *docile* [DG79].

In first-order logic *without equality*, there are three docile *prefix classes*, i.e., classes of first-order sentences defined by their quantifier prefix [BGG97]:

- The *Bernays-Schönfinkel class*, which is the collection of all first-order sentences with prefixes of the form $\exists^*\forall^*$ (i.e., the existential quantifiers precede the universal quantifiers).

- The *Ackermann class*, which is the collection of all first-order sentences with prefixes of the form $\exists^*\forall\exists^*$ (i.e., the prefix contains a single universal quantifier).
- The *Gödel class*, which is the collection of all first-order sentences with prefixes of the form $\exists^*\forall\forall\exists^*$ (i.e., the prefix contains two consecutive universal quantifiers).

These three classes are also the only prefix classes that have a solvable satisfiability problem [BGG97]. In first-order logic *with equality*, the Gödel class is not docile and its satisfiability problem is not solvable [Gol84]. This is the only class where equality makes a difference.

We focus here on the question whether the 0-1 law holds for the Σ_1^1 fragments defined by first-order prefix classes, and whether or not the associated decision problem for the probabilities is solvable. This can be viewed as a classification of the prefix classes according to whether the corresponding Σ_1^1 fragments have a 0-1 law. This classification project was launched in [KV87] and was completed only recently in [LeB98]. For first-order logic with equality, the classifications of prefix classes according to their docility, i.e., according to the solvability of their finite satisfiability problem, and according to the 0-1 law for the corresponding Σ_1^1 fragment are identical. Moreover, 0-1 laws in this classification are always accompanied by solvability of the decision problem for the probabilities. This is manifested by the positive results for the classes Σ_1^1 (Bernays-Schönfinkel) and Σ_1^1 (Ackermann), and the negative results for the other classes. For first-order logic with equality, the two classification differ, as the 0-1 law fails for the class Σ_1^1 (Gödel) and the association classification problem is undecidable.

This paper is a survey that focuses on the overall picture rather than on technical details. The interested reader is referred to the cited papers for further details. Our main focus here is on positive results involving 0-1 laws. For a survey that focus on negative results, see [LeB00]. For an earlier overview, which includes a focus on expressiveness issues, see [KV89]. See [Lac97] for results on 0-1 laws for second-order fragments that involves alternation of second-order quantifiers.

2 Random Structures

Let $\mathbf{R}$ be a vocabulary consisting of relation symbols *only* and let C be the collection of all finite relational structures over $\mathbf{R}$ whose universes are initial segments $\{1, 2, \dots, n\}$ of the integers. If P is a property of (some) structures in C , then let $\mu_n(P)$ be the fraction of structures in C of cardinality n satisfying P . The *asymptotic probability* $\mu(P)$ on C is defined to be $\mu(P) = \lim_{n \rightarrow \infty} \mu_n(P)$, provided this limit exists. In this probability space all structures in C with the same number of elements carry the same probability. An equivalent description of this space can be obtained by assigning truth values to tuples independently and with the same probability (cf. [Bol85]).

If L is a logic, we say that the *0-1 law holds for L on C* in case $\mu(P)$ exists and is equal to 0 or 1 for every property P expressible in the logic L . We write $\Theta(L)$ for the collection of all sentences P in L with $\mu(P) = 1$. Notice that if L is first-order logic, then the existence of the 0-1 law is equivalent to stating that $\Theta(L)$ is a complete theory.

A standard method for establishing 0-1 laws, originating in Fagin [Fag76], is to prove that the following *transfer theorem* holds: there is an infinite structure $\mathbf{A}$ over the vocabulary $\mathbf{R}$ such that for every property P expressible in L we have: $\mathbf{A} \models P \iff$

$\mu(P) = 1$. It turns out that there is a unique (up to isomorphism) countable structure $\mathbf{A}$ that satisfies the above equivalence for first-order logic and for the fragments of second-order logic considered here. We call $\mathbf{A}$ the *countable random structure over the vocabulary $\mathbf{R}$* . The structure $\mathbf{A}$ is characterized by an infinite set of *extension axioms*, which, intuitively, assert that every *type* can be *extended* to every other possible type. More precisely, if $\mathbf{x} = (x_1, \dots, x_n)$ is a sequence of variables, then a n - $\mathbf{R}$ -type $t(\mathbf{x})$ in the variables $\mathbf{x}$ over $\mathbf{R}$ is a maximal consistent set of equality and negated equality formulas and atomic and negated atomic formulas from the vocabulary $\mathbf{R}$ in the variables $x_1, \dots, x_n$. We say that a $(n+1)$ - $\mathbf{R}$ -type $s(\mathbf{x}, z)$ *extends the type $t(\mathbf{x})$* if t is a subset of s . Every type $t(\mathbf{x})$ can be also viewed as a quantifier-free formula that is the conjunction of all members of $t(\mathbf{x})$. With each pair of types s and t such that s extends t we associate a first-order *extension axiom* τ which states that $(\forall \mathbf{x})(t(\mathbf{x}) \rightarrow (\exists z)s(\mathbf{x}, z))$.

Let T be the set of all extension axioms. The theory T was studied by Gaifman [Gai64], who showed, using a *back and forth* argument, that every two countable models of T are isomorphic (i.e., T is an ω -categorical theory). The extension axioms can also be used to show that the unique (up to isomorphism) countable model $\mathbf{A}$ of T is *universal* for all countable structures over $\mathbf{R}$, i.e., if $\mathbf{B}$ is a countable structure over $\mathbf{R}$, then there is a substructure of $\mathbf{A}$ that is isomorphic to $\mathbf{B}$.

Fagin [Fag76] realized that the extension axioms are relevant to the study of probabilities on finite structures and proved that on the class C of all finite structures over a vocabulary $\mathbf{R}$ $\mu(\tau) = 1$ for every extension axiom τ . The 0-1 law for first-order logic and the transfer theorem between truth of first-order sentences on $\mathbf{A}$ and almost sure truth of such sentences on C follows from these results by a compactness argument. We should point out that there are different proofs of the 0-1 law for first-order logic, which have a more elementary character (cf. [GKLT69, Com88]). These proofs do not deploy infinite structures or the compactness theorem and they bypass the transfer theorem. In contrast, the proofs of the 0-1 laws for fragments of second-order logic that we present here do involve infinitistic methods. Lacoste showed how these infinitistic arguments can be avoided [Lac96].

Since the set T of extension axioms is recursive, it also follows that $\Theta(L)$ is recursive, where L is first-order logic. In other words, there is an algorithm to decide the value (0 or 1) of the asymptotic probability of every first-order sentence. The computational complexity of this decision problem was investigated by Grandjean [Gra83], who showed that it is PSPACE-complete, when the underlying vocabulary $\mathbf{R}$ is assumed to be *bounded* (i.e., there is a some bound on the arity of the relation symbols in σ).

3 Existential and Universal Second-Order Sentences

The Σ_1^1 and Π_1^1 formulas form the syntactically simplest fragment of second-order logic. A Σ_1^1 formula over a vocabulary $\mathbf{R}$ is an expression of the form $(\exists \mathbf{S})\theta(\mathbf{S})$, where $\mathbf{S}$ is a sequence of relation symbols not in the vocabulary $\mathbf{R}$ and $\theta(\mathbf{S})$ is a first-order formula over the vocabulary $\mathbf{R} \cup \mathbf{S}$. A Π_1^1 formula is an expression of the form $(\forall \mathbf{S})\theta(\mathbf{S})$, where $\mathbf{S}$ and $\theta(\mathbf{S})$ are as above.

Both the 0-1 law and the transfer theorem fail for arbitrary Σ_1^1 and Π_1^1 sentences. Consider, for example, the statement “there is relation that is the graph of a permutation in

which every element is of order 2". On finite structures this statement is true exactly when the universe of the structure has an even number of elements and, as a result, it has no asymptotic probability. This statement, however, is expressible by a Σ_1^1 sentence, which, moreover, is true on the countable random structure $\mathbf{A}$. Similarly, the statement "there is a total order with no maximum element" is true on the countable random structure $\mathbf{A}$, but is false on every finite structure. Notice that in the two preceding examples the transfer theorem for Σ_1^1 sentences fails in the direction from truth on the countable random structure $\mathbf{A}$ to almost sure truth on finite structures. In contrast, the following simple lemma shows that this direction of the transfer theorem holds for all Π_1^1 sentences.

Lemma 1. [KV87] *Let $\mathbf{A}$ be the countable random structure over $\mathbf{R}$ and let $(\forall \mathbf{S})\theta(\mathbf{S})$ be an arbitrary Π_1^1 sentence. If $\mathbf{A} \models (\forall \mathbf{S})\theta(\mathbf{S})$, then there is a first order sentence ψ over the vocabulary σ such that: $\mu(\psi) = 1$ and $\models \psi \rightarrow (\forall \mathbf{S})\theta(\mathbf{S})$. In particular, every Π_1^1 sentence that is true on $\mathbf{A}$ has probability 1 on C .*

The proof of Lemma 1 uses the Compactness Theorem. For an approach that avoid the usage of infinitistic arguments, see [Lac96].

Corollary 1. [KV87] *Every Σ_1^1 sentence that is false on the countable random structure $\mathbf{A}$ has probability 0 on C .*

Corollary 2. [KV87] *The set of Π_1^1 sentences that are true on $\mathbf{A}$ is recursively enumerable.*

Proof: It shown in [KV87] that $\mathbf{A} \models (\forall \mathbf{S})\theta(\mathbf{S})$ iff $(\forall \mathbf{S})\theta(\mathbf{S})$ is logically implied by the set T of extension axioms. ■

We investigate here classes of Σ_1^1 and Π_1^1 sentences that are obtained by restricting appropriately the pattern of the first-order quantifiers in such sentences. If $\mathcal{F}$ is a class of first-order formulas, then we write $\Sigma_1^1(\mathcal{F})$ for the collection of all Σ_1^1 sentences whose first-order part is in $\mathcal{F}$.

The discussion in the introduction suggests that we consider prefix classes $\mathcal{F}$ that are docile, i.e., they have a solvable finite satisfiability problem. Thus, we focus on the following classes of existential second-order sentences:

- The class $\Sigma_1^1(\exists^*\forall^*)$ of Σ_1^1 sentences whose first-order part is a *Bernays-Schönfinkel* formula.
- The class $\Sigma_1^1(\exists^*\forall\exists^*)$ of Σ_1^1 sentences whose first-order part is an *Ackermann* formula.
- The class $\Sigma_1^1(\exists^*\forall\forall\exists^*)$ of Σ_1^1 sentences whose first-order part is a *Gödel* formula.

We also refer to the above as the Σ_1^1 (Bernays-Schönfinkel) class, the Σ_1^1 (Ackermann) class, and the Σ_1^1 (Gödel) class, respectively. We consider these classes both with and without equality.

Fagin [Fag74] showed that a class of finite structures over a vocabulary $\mathbf{R}$ is NP computable if and only if it is definable by a Σ_1^1 sentence over $\mathbf{R}$. The restricted classes of Σ_1^1 sentences introduced above can not express all NP problems on finite structures. In spite of their syntactic simplicity, however, the classes $\Sigma_1^1(\exists^*\forall^*)$, $\Sigma_1^1(\exists^*\forall\exists^*)$ and $\Sigma_1^1(\exists^*\forall\forall\exists^*)$ can express natural NP-complete problems [KV87,KV90].

4 The Class Σ_1^1 (Bernays-Schönfinkel) with Equality

4.1 0-1 Law

Lemma 1 and Corollary 1 reveal that in order to establish the 0-1 law for a class $\mathcal{F}$ of existential second-order sentences it is enough to show that if Ψ is a sentence in $\mathcal{F}$ that is true on the countable random structure $\mathbf{A}$, then $\mu(\Psi) = 1$. In this section we prove this to be the case for the class of Σ_1^1 (Bernays-Schönfinkel) sentences.

Lemma 2. [KV87] *Let $(\exists \mathbf{S})(\exists \mathbf{x})(\forall \mathbf{y})\theta(\mathbf{S}, \mathbf{x}, \mathbf{y})$ be a $\Sigma_1^1(\exists^*\forall^*)$ sentence that is true on the countable random structure $\mathbf{A}$. Then there is a first order sentence ψ over σ such that $\mu(\psi) = 1$ and $\models_{fin} \psi \rightarrow (\exists \mathbf{S})(\exists \mathbf{x})(\forall \mathbf{y})\theta(\mathbf{S}, \mathbf{x}, \mathbf{y})$, where $\models_{fin}$ denotes truth in all finite structures. In particular, if Ψ is a $\Sigma_1^1(\exists^*\forall^*)$ sentence that is true on $\mathbf{A}$, then $\mu(\Psi) = 1$.*

Proof: Let $\mathbf{a} = (a_1, \dots, a_n)$ be a sequence of elements of $\mathbf{A}$ that witness the first-order existential quantifiers $\mathbf{x}$ in $\mathbf{A}$. Let $\mathbf{A}_0$ be the finite substructure of $\mathbf{A}$ with universe $\{a_1, \dots, a_n\}$. Then there is a first-order sentence ψ , which is the conjunction of a finite number of the extension axioms, having the property that every model of it contains a substructure isomorphic to $\mathbf{A}_0$. Now assume that $\mathbf{B}$ is a finite model of ψ . Using the extension axioms we can find a substructure $\mathbf{B}^*$ of the random structure $\mathbf{A}$ that contains $\mathbf{A}_0$ and is isomorphic to $\mathbf{B}$. Since universal statements are preserved under substructures, we conclude that $\mathbf{B} \models (\exists \mathbf{S})(\exists \mathbf{x})(\forall \mathbf{y})\theta(\mathbf{S}, \mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ is interpreted by $\mathbf{a}$ and $\mathbf{S}$ is interpreted by the restriction to $\mathbf{B}$ of the relations on $\mathbf{A}$ that witness the existential second-order quantifiers. ■

From Lemmas 1 and 2 we infer immediately the 0-1 law and the transfer theorem for the class $\Sigma_1^1(\exists^*\forall^*)$.

Theorem 1. [KV87] *The 0-1 law holds for Σ_1^1 (Bernays-Schönfinkel) sentences on the class $\mathcal{C}$ of all finite structures over a relational vocabulary $\mathbf{R}$. Moreover, if $\mathbf{A}$ is the countable random structure and Ψ is a Σ_1^1 (Bernays-Schönfinkel) sentence, then $\mathbf{A} \models \Psi \iff \mu(\Psi) = 1$.*

4.2 Solvability

As mentioned in Section 2, the proof of the 0-1 law for first-order logic showed also the solvability of the decision problem for the values (0 or 1) of the probabilities of first-order sentences. The preceding proof of the 0-1 law for Σ_1^1 (Bernays-Schönfinkel) sentences does not yield a similar result for the associated decision problem for the probabilities of such sentences. Indeed, the only information one can extract from the proof is that the Σ_1^1 (Bernays-Schönfinkel) sentences of probability 0 form a recursively enumerable set. We now show that the decision problem for the probabilities of sentences in the class Σ_1^1 (Bernays-Schönfinkel) is solvable. We do this by proving that satisfiability of such sentences on $\mathbf{A}$ is equivalent to the existence of certain *canonical* models. For simplicity we present the argument for $\Sigma_1^1(\forall^*)$ sentences, i.e., sentences of the form

$$\exists S_1 \dots \exists S_l \forall y_1 \dots \forall y_m \theta(S_1, \dots, S_l, y_1, \dots, y_m).$$

Assume that the vocabulary σ consists of a sequence $\mathbf{R} = \langle R_i, i \in I \rangle$ of relation variables R_i . If B is a set and, for each $i \in I$, R_i^B is a relation on B of the same arity as that of R_i , then we write $\mathbf{R}^B$ for the sequence $\langle R_i^B, i \in I \rangle$. Let $<$ be a new binary relation symbol and consider structures $\mathbf{B} = (B, \mathbf{R}^B, <^B)$ in which $<^B$ is a *total ordering*. Let k be a positive integer. We say that $\mathbf{B}$ is *k-rich* if for every structure $\mathbf{D}$ with k elements over $\mathbf{R} \cup \{<\}$ (where $<$ is interpreted by a total ordering) there is a substructure $\mathbf{B}^*$ of $\mathbf{B}$ that is isomorphic to $\mathbf{D}$. Notice that the isomorphism takes into account *both* the relations R_i^D and the total ordering $<^D$.

Assume that S^B is an n -ary relation on B . We say that S^B is *canonical for the structure* $\mathbf{B} = (B, \mathbf{R}^B, <^B)$ if for every sequence $\mathbf{b} = (b_1, \dots, b_n)$ from B the truth value of $S^B(b_1, \dots, b_n)$ depends *only* on the isomorphism type of the substructure of $\mathbf{B}$ with universe $\{b_1, \dots, b_n\}$. An expanded structure $\mathbf{B}^* = (B, \mathbf{R}^B, <^B, S_1^B, \dots, S_l^B)$ is *canonical* if every relation S_i^B , $1 \leq i \leq l$, is canonical on $\mathbf{B}$. The intuition behind canonical structures is that the relations S_i^B are determined completely by the $(\mathbf{R}, <)$ -types.

We can state now the main technical result of this section.

Theorem 2. [KV87] *Let $\mathbf{A}$ be the countable random structure over the vocabulary σ and let Ψ be a Σ_1^1 sentence of the form $\exists S_1 \dots \exists S_l \forall y_1 \dots \forall y_m \theta(S_1, \dots, S_l, y_1, \dots, y_m)$. Then the following are equivalent:*

1. $\mathbf{A} \models \Psi$.
2. *There is a finite canonical structure $\mathbf{B}^* = (B, \mathbf{R}^B, <^B, S_1^B, \dots, S_l^B)$ which is k -rich for every $k \leq m$ and such that $\mathbf{B}^* \models \forall y_1 \dots \forall y_m \theta(S_1, \dots, S_l, y_1, \dots, y_m)$.*

In showing that (1) $\implies$ (2) we will use certain Ramsey-type theorems that were proved by [NR77, NR83] and independently by [AH78]. We follow here the notation and terminology of [AH78] in stating these combinatorial results.

If X is a set and j is an integer, then $[X]^j$ is the collection of all subsets of X with j elements and $[X]^{\leq n} = \bigcup_{j \leq n} [X]^j$. A *system of colors* is a sequence $\mathbf{K} = (K_1, \dots, K_n)$ of finite nonempty sets. A *$\mathbf{K}$ -colored set* consists of a finite set X , a (total) ordering $<^X$ on X and a function $f : [X]^{\leq n} \mapsto K_1 \cup \dots \cup K_n$ such that $f(Z) \in K_j$ for every $Z \in [X]^j$ and $j \leq n$.

It is clear that every $\mathbf{K}$ -colored set is isomorphic to a unique *$\mathbf{K}$ -pattern*, that is a $\mathbf{K}$ -colored set whose underlying set is an integer. If e, M are integers, $\mathbf{K}$ is a system of colors, P, Q are $\mathbf{K}$ -patterns, then $Q \hookrightarrow (P)_{\leq M}^e$ means that for every $\mathbf{K}$ -colored set (X, f) of pattern Q and every partition $F : (X)^{\leq e} \mapsto M$ there is a subset Y of X such that $(Y, f|_Y)$ is of pattern P , and Y is *conditionally monochromatic* for F as $\mathbf{K}$ -colored set, i.e. for $Z \in [Y]^j$ the value $F(Z)$ depends *only* on the $\mathbf{K}$ -pattern of $(Z, f|_Z)$.

By iterated applications of Theorem 2.2 in [AH78], we can derive the following generalization of the classical Ramsey theorem [Ram28]:

Theorem 3. [KV87] *For arbitrary integers e, M , a system of colors $\mathbf{K}$, and a $\mathbf{K}$ -pattern P , there is a $\mathbf{K}$ -pattern Q such that $Q \hookrightarrow (P)_{\leq M}^e$.*

With every finite vocabulary $\mathbf{R}$ in which the maximum arity is n we can associate a system of colors $\mathbf{K}$ such that every finite structure $\mathbf{B} = (B, \mathbf{R}^B, <^B)$, where $<^B$ is a total ordering on B , can be *coded* by a $\mathbf{K}$ -colored set $(B, <^B, f)$ with $f : [B]^{\leq n} \mapsto$

$K_1 \cup \dots \cup K_n$. For example, if σ consists of a single binary relation R , then K_1 has 2 elements, K_2 has 4 elements, and $f : [B]^{\leq 2} \mapsto K_1 \cup K_2$ is such that the value $f(\{x\})$ depends only on the truth value of $R^B(x, x)$, while the value $f(\{x, y\})$ depends only on the truth values of $R^B(\min(x, y), \max(x, y))$ and $R^B(\max(x, y), \min(x, y))$. Conversely, from every such $\mathbf{K}$ -pattern we can *decode* a finite structure $\mathbf{B}$.

We now have all the combinatorial machinery needed to outline the ideas in the proof of Theorem 2.

Sketch of Proof of Theorem 2.

(1) $\implies$ (2) Let Ψ be the Σ_1^1 sentence Ψ such that $\mathbf{A} \models \Psi$ and assume for simplicity that Ψ has only one ternary second-order existential variable S . We use the ternary relation S^A witnessing S on $\mathbf{A}$ to partition $[A]^{\leq 3}$ according to the *pure* S -type of a set $Z \in [A]^{\leq 3}$. This means that A is partitioned into two pieces defined by the truth value of $S^A(x, x, x)$, $[A]^2$ into $2^{2^3-2} = 64$ pieces defined by the truth values of $S^A(\min(x, y), \max(x, y), \max(x, y))$, etc., and finally $[A]^3$ is partitioned into $2^{3!} = 64$ pieces defined by the truth values of $S^A(\min(x, y, z), \text{mid}(x, y, z), \max(x, y, z))$, etc..

Let $\mathbf{B} = (B, \mathbf{R}^B, <^B)$ be a finite structure which is k -rich for every $k \leq m$ and let P be a $\mathbf{K}$ -pattern which codes $\mathbf{B}$ in the way described above. We apply now Theorem 3 for $e = 3$, $M = 2 + 64 + 64 = 130$, and for P coding $\mathbf{B}$. Let Q be a $\mathbf{K}$ -pattern such that $Q \hookrightarrow (P)_M^{\leq 3}$ and let $\mathbf{C}$ be the structure coded by Q . Using the extension axioms for the countable random structure $\mathbf{A}$ we can find in $\mathbf{A}$ a substructure $\mathbf{C}_1$ isomorphic to $\mathbf{C}$. But now Theorem 3 guarantees that $\mathbf{C}_1$ contains a substructure $\mathbf{B}_1$ which is isomorphic to $\mathbf{B}$ and is conditionally monochromatic as a $\mathbf{K}$ -colored pattern. The structure $\mathbf{B}_1$ is k -rich for every $k \leq m$ and by taking the restriction of S^A on B_1 we can expand $\mathbf{B}_1$ to a canonical model $\mathbf{B}^*$ of $\forall y_1 \dots \forall y_m \theta(S, y_1, \dots, y_m)$, since universal sentences are preserved under substructures.

(2) $\implies$ (1) From every canonical, k -rich model ($1 \leq k \leq m$) of

$$\forall y_1 \dots \forall y_m \theta(S, y_1, \dots, y_m)$$

we can build a relation S^A witnessing the second order existential quantifier in Ψ by assigning tuples to S^A according to their $(\mathbf{R}, <)$ -type. ■

Theorem 2 implies that the set of $\Sigma_1^1(\exists^* \forall^*)$ properties having probability 1 is recursively enumerable. On the other hand, Theorem 1 and Corollary 2 together imply that the complement of this set is also recursively enumerable. Thus, we have established that the decision problem for the probabilities of strict Σ_1^1 properties is solvable. This proof does not give, however, any complexity bounds for the problem. In [KV87] we analyzed the computational complexity of this decision problem and showed that it is NEXPTIME-complete for bounded vocabularies and 2NEXPTIME for unbounded vocabularies.

5 The Class $\Sigma_1^1(\text{Ackermann})$ with Equality

Our goal here is to establish the following:

Theorem 4. [KV90] *Let $\mathbf{A}$ be the countable random structure over the vocabulary $\mathbf{R}$ and let Ψ be a $\Sigma_1^1(\text{Ackermann})$ sentence. If $\mathbf{A} \models \Psi$, then $\mu(\Psi) = 1$.*

This theorem will be obtained by combining three separate lemmas. Since the whole argument is rather involved, we start with a “high-level” description of the structure of the proof.

We first isolate a syntactic condition (condition (χ) below) for Σ_1^1 (Ackermann) sentences and in Lemma 3 we show that if Ψ is a Σ_1^1 (Ackermann) sentence which is true on $\mathbf{A}$, then condition (χ) holds for Ψ . At the end, it will actually turn out that this condition (χ) is also sufficient for truth of Σ_1^1 (Ackermann) sentences on the countable random structure $\mathbf{A}$. In Lemma 4, we isolate a “richness” property E_s , $s \geq 1$, of (some) finite structures over $\mathbf{R}$ and show that $\mu(E_s) = 1$ for every $s \geq 1$. The proof of this lemma requires certain asymptotic estimates from probability theory, due to Chernoff [Che52]. Finally, in Lemma 5, we prove that if Ψ is a Σ_1^1 (Ackermann) sentence for which condition (χ) holds, then for appropriately chosen s and for all large n the sentence Ψ is true on all finite structures of cardinality n over $\mathbf{R}$ that possess property E_s ; consequently, $\mu(\Psi) = 1$. In this last lemma, the existence of the predicates $\mathbf{S}$ that witness Ψ is proved by a probabilistic argument, which in spirit is analogous to the technique used by Gurevich and Shelah [GS83] for showing the finite satisfiability property of first-order formulas in the Gödel class without equality.

Let $\mathbf{T}$ be a vocabulary, i.e. a set of relational symbols. Recall that, a k - $\mathbf{T}$ -type $t(x_1, \dots, x_k)$ is a maximal consistent set of equality, negated equality formulas, atomic and negated atomic formulas whose variables are among $x_1, \dots, x_k$.

- If $t(x_1, \dots, x_k)$ is a k - $\mathbf{T}$ -type, then, for every m with $1 \leq m \leq k$, let $t(x_{i_1}, \dots, x_{i_m})$ be the m - $\mathbf{T}$ -type obtained by deleting from $t(x_1, \dots, x_k)$ all formulas in which a variable $y \neq x_{i_1}, \dots, x_{i_m}$ occurs.
- If $\mathbf{S} \subseteq \mathbf{T}$, then the *restriction* of t to $\mathbf{S}$ is the k - $\mathbf{S}$ -type obtained by deleting from t all formulas in which a predicate symbols in $\mathbf{T} - \mathbf{S}$ occurs.
- If $t(x_1, \dots, x_k, x_{k+1})$ is a $(k+1)$ - $\mathbf{T}$ -type, and y is a variable different from all the x_i ’s, then $t(x_1, \dots, x_k, x_{k+1}/y)$ is a $(k+1)$ - $\mathbf{T}$ -type obtained by replacing all occurrences of x_{k+1} by y .
- Let $t(x_1, \dots, x_k)$ be a k - $\mathbf{T}$ -type, and let $\phi(x_1, \dots, x_k)$ be a quantifier-free formula in the variables $x_1, \dots, x_k$. We say that t *satisfies* ϕ if ϕ is true under the truth assignment that assigns true to an atomic formula precisely when it is a member of t .

Let Ψ be a Σ_1^1 (Ackermann) sentence of the form

$$(\exists \mathbf{S})(\exists x_1) \dots (\exists x_k)(\forall y)(\exists z_1) \dots (\exists z_l)\phi(x_1, \dots, x_k, y, z_1, \dots, z_l, \mathbf{R}, \mathbf{S}),$$

where ϕ is a quantifier-free formula over the vocabulary $\mathbf{R}, \mathbf{S} = \mathbf{R} \cup \mathbf{S}$.

We say that *condition (χ) holds for Ψ* if there is k - $(\mathbf{R}, \mathbf{S})$ -type $t_0(x_1, \dots, x_k)$ and a set P of $(k+1)$ - $(\mathbf{R}, \mathbf{S})$ -types $t(x_1, \dots, x_k, y)$ extending $t_0(x_1, \dots, x_k)$ such that the following are true:

1. P contains as a member the $(k+1)$ - $(\mathbf{R}, \mathbf{S})$ -type $t_0^{x_i}(x_1, \dots, x_k, y)$, for every $i = 1 \dots k$. Equivalently, for every i , $1 \leq i \leq k$, there is a type $t_i(x_1, \dots, x_k, y)$ in P such that $t_i(x_1, \dots, x_k, y/x_i) = t_0(x_1, \dots, x_k)$.
2. P is $\mathbf{R}$ -rich over $t_0(x_1, \dots, x_k)$, i.e., every $(k+1)$ - $\mathbf{R}$ -type $t(x_1, \dots, x_k, y)$ extending the restriction of $t_0(x_1, \dots, x_k)$ to $\mathbf{R}$ is itself the restriction of some $(k+1)$ - $(\mathbf{R}, \mathbf{S})$ -type in P to $\mathbf{R}$.

3. For each $t(x_1, \dots, x_k, y)$ in P there is a $(k + l + 1)$ -($\mathbf{R}, \mathbf{S}$)-type

$$t'(x_1, \dots, x_k, y, z_1, \dots, z_l)$$

such that $t \subseteq t'$, t' satisfies $\phi(x_1, \dots, x_k, y, z_1, \dots, z_l)$, and for each z_i , $1 \leq i \leq l$, the $(k + 1)$ -($\mathbf{R}, \mathbf{S}$)-type $t'(x_1, \dots, x_k, z_i/y)$ is in P .

Lemma 3. [KV90] *Let $\mathbf{A}$ be the countable random structure over the vocabulary $\mathbf{R}$ and let Ψ be a Σ_1^1 (Ackermann) sentence. If $\mathbf{A} \models \Psi$, then condition (χ) holds for Ψ .*

Proof: (Hint) The type t_0 and the set of types P required in condition (χ) are obtained from the relations on $\mathbf{A}$ and the elements of $\mathbf{A}$ that witness the existential second-order quantifiers $(\exists \mathbf{S})$ and the existential first-order quantifiers $(\exists x_1) \dots (\exists x_k)$ in Ψ respectively. To show that P is $\mathbf{R}$ -rich, we use the fact that the countable random structure $\mathbf{A}$ satisfies the extension axioms, which in turn imply that the elements of $\mathbf{A}$ realize all possible $\mathbf{R}$ -types. ■

Let D be a structure over $\mathbf{R}$ and let $\bar{c} = (c_1, \dots, c_m)$ be a sequence of elements from D . The type $t_{\bar{c}}$ of $\bar{c}$ on D is the unique m - $\mathbf{R}$ -type $t(z_1, \dots, z_m)$ determined by the atomic and negated atomic formulas that the sequence $\bar{c}$ satisfies on D , under the assignment $z_i \rightarrow c_i$, $1 \leq i \leq m$. We say that a sequence $\bar{c}$ realizes a type t on a structure D if $t_{\bar{c}} = t$.

Let $s \geq 1$ be fixed. We say that a finite structure D over $\mathbf{R}$ with n elements satisfies property E_s if the following holds:

- For every number m with $1 \leq m \leq s$, every sequence $\bar{c} = (c_1, \dots, c_m)$ from D and every $(m + 1)$ - $\mathbf{R}$ -type $t(z_1, \dots, z_m, z_{m+1})$ extending the type $t_{\bar{c}}$ of $\bar{c}$ on D , there are at least $\sqrt{n}$ different elements d in D such that each sequence $(c_1, \dots, c_m, d)$ realizes the type $t(z_1, \dots, z_m, z_{m+1})$.

Lemma 4. [KV90] *For every $s \geq 1$ there is a positive constant c and a natural number n_0 such that for every $n \geq n_0$ $\mu_n(E_s) \geq 1 - n^{s+1}e^{-cn}$. In particular, $\mu(E_s) = 1$, i.e. almost all structures over $\mathbf{R}$ satisfy property E_s , for every $s \geq 1$.*

Proof: (Sketch) The proof of this lemma uses an asymptotic bound on the probability in the tail of the binomial distribution, due to Chernoff [Che52] (cf. also [Bol85]). We first fix a sequence $\bar{c}$ from D and a type t that extends $t_{\bar{c}}$, and apply this bound to the binomial distribution obtained by counting the number of elements d such that the sequence $(c_1, \dots, c_m, d)$ realizes t . We then iterate through all types and all sequences $\bar{c} = (c_1, \dots, c_m)$ for $1 \leq m \leq s$. ■

The last lemma in this section provides the link between condition (χ) , property E_s , $s \geq 1$, and satisfiability of Σ_1^1 (Ackermann) sentences on finite structures over $\mathbf{R}$.

Lemma 5. [KV90] *Let Ψ be a Σ_1^1 (Ackermann) sentence of the form*

$$(\exists \mathbf{S})(\exists x_1) \dots (\exists x_k)(\forall y)(\exists z_1) \dots (\exists z_l)\phi(x_1, \dots, x_k, y, z_1, \dots, z_l, \mathbf{R}, \mathbf{S})$$

for which condition (χ) holds. There is a natural number n_1 such that for every $n \geq n_1$, if D is a finite structure over $\mathbf{R}$ with n elements satisfying property E_{k+l+1} , then $D \models \Psi$.

Proof: (Sketch) The existence of the relations on D that witness the second-order quantifiers $(\exists \mathbf{S})$ in Ψ is proved with a probabilistic argument similar to the one employed by Gurevich and Shelah [GS83] for the finite satisfiability property of the Gödel class without equality. We use condition (χ) to impose on D a probability space of $\mathbf{S}$ predicates. The richness property E_{k+l+1} is then used to show that with nonzero probability (in this new space) the expansion of D with these predicates satisfies the sentence $(\exists x_1) \dots (\exists x_k)(\forall y)(\exists z_l) \dots (\exists z_l)\phi(y, z_1, \dots, z_l, \mathbf{S})$. ■

This completes the outline of the proof of Theorem 4. Combining now this theorem with Lemma 1 we derive the main result of this section.

Theorem 5. [KV90] *The 0-1 law holds for the $\Sigma_1^1(\text{Ackermann})$ class on the collection C of all finite structures over a vocabulary $\mathbf{R}$. Moreover, if $\mathbf{A}$ is the countable random structure over $\mathbf{R}$ and Ψ is a $\Sigma_1^1(\text{Ackermann})$ sentence, then $\mathbf{A} \models \Psi \iff \mu(\Psi) = 1$.*

Notice that the preceding results also show that a $\Sigma_1^1(\text{Ackermann})$ sentence Ψ has probability 1 if and only if condition (χ) holds for Ψ . Since condition (χ) is clearly effective, it follows that the decision problem for the values of the probabilities of $\Sigma_1^1(\text{Ackermann})$ sentences is solvable. In [KV90] we analyzed the computational complexity of this decision problem and showed that it is NEXPTIME-complete for bounded vocabularies and Σ_2^{exp} -complete¹ for unbounded vocabularies.

6 Negative Results and Classifications

The Bernays-Schönfinkel and Ackermann classes are the only docile prefix classes with equality, i.e., they are the only prefix classes of first-order logic with equality for which the finite satisfiability problem is solvable [BGG97]. A key role in this classification was played by the *Gödel class* with equality, which is the class of first-order sentences with equality and with prefix of the form $\forall\forall\exists^*$. In fact, the classification was completed only when Goldfarb [Gol84] showed that the *minimal Gödel class*, i.e., the class of first-order sentences with equality and with prefix of the form $\forall\forall\exists$, is not docile (contradicting an unproven claim in [God32]). We now show that in the presence of equality the same classification holds for the 0-1 law, namely, the 0-1 law holds for the Σ_1^1 fragments that correspond to docile prefix classes.

It is easy to see that the 0-1 law does not hold for the Σ_1^1 fragments that correspond to the prefix classes $\forall\exists\forall$ and $\forall^3\exists$. For example, the PARITY property, i.e., the property “there is an even number of elements” can be expressed by the following $\Sigma_1^1(\forall\exists\forall)$ sentence asserting that “there is a permutation in which every element is of order 2”:

$$(\exists S)(\forall x)(\exists y)(\forall z)[S(x, y) \wedge (S(x, z) \rightarrow y = z) \wedge (S(x, z) \leftrightarrow S(z, x)) \wedge \neg S(x, x)].$$

¹ Σ_2^{exp} is the second-level of the exponential hierarchy. It can be described as the class of languages accepted by alternating exponential-time Turing machines in two alternations where the machine start state is existential [CKS81] or as the class NEXP^{NP} of languages accepted by nondeterministic exponential-time Turing machines with oracles from NP [HIS85].

The statement “there is a permutation in which every element is of order 2” can also be expressed by the following $\Sigma_1^1(\forall\forall\forall\exists)$ sentence

$$(\exists S)(\forall x)(\forall y)(\forall z)(\exists w)[S(x, w) \wedge (S(x, y) \wedge S(x, z) \rightarrow y = z) \wedge (S(x, z) \leftrightarrow S(z, x)) \wedge \neg S(x, x)].$$

Dealing with the class $\Sigma_1^1(\text{Gödel})$, i.e., the class $\Sigma_1^1(\forall\forall\exists^*)$ is much harder.

Theorem 6. [PS91,PS93] *The 0-1 law fails for the class $\Sigma_1^1(\forall\forall\exists)$.*

Proof: (*Sketch*): The proof proceed by construting a $\Sigma_1^1(\forall\forall\exists)$ sentence Ψ (with 43 clauses!) over a certain vocabulary $\mathbf{R}$ such that a finite structure D over $\mathbf{R}$ satisfies Ψ if and only if the cardinality of the universe of D is of the form $(n^2 + 3n + 4)/2$ for some integer n .

The construction of the above sentence Ψ uses ideas from Goldfarb’s [Gol84] proof of the unsolvability of the satisfiability problem for the Gödel class. The main technical innovation in that proof was the construction of a $\forall\forall\exists$ first-order sentence ϕ that is satisfiable, but has no finite models. The $\Sigma_1^1(\text{Gödel})$ sentence Ψ that has no asymptotic probability is obtained by modifying ϕ appropriately. ■

We can conclude that for first-order logic with equality the classifications of prefix classes according to their docility and according to the 0-1 law for the corresponding Σ_1^1 fragments are identical. This follows from the positive results for the classes $\Sigma_1^1(\text{Bernays-Schönfinkel})$ and $\Sigma_1^1(\text{Ackermann})$, and the negative results for the classes $\Sigma_1^1(\forall\exists\forall)$, $\Sigma_1^1(\forall^3)$, and $\Sigma_1^1(\forall\forall\exists)$.

Let us now consider the classification for the prefix classes without equality. Clearly, the 0-1 laws for the classes $\Sigma_1^1(\text{Bernays-Schönfinkel})$ and $\Sigma_1^1(\text{Ackermann})$ hold. On the other hand, the sentences used to demonstrate the failure of the 0-1 laws for $\Sigma_1^1(\forall\exists\forall)$, $\Sigma_1^1(\forall^3\exists)$, and $\Sigma_1^1(\forall\forall\exists)$ all used equality. To complete the classification without equality, we need to settle the status of the 0-1 law for the equality-free version of the latter three classes.

Consider first the class $\Sigma_1^1(\forall^3\exists)$. We showed earlier that it can express PARITY using equality. It turns out that without equality it can express PARITY almost surely. Consider the sentence

$$(\exists R)(\exists S)(\forall x)(\forall y)(\forall z)(\exists w)[S(x, w) \wedge (S(x, y) \wedge S(x, z) \rightarrow R(y, z)) \wedge (S(x, z) \leftrightarrow S(z, x)) \wedge \neg S(x, x) \wedge (R(x, y) \leftrightarrow (E(x, z) \leftrightarrow E(y, z)))].$$

It is shown in [KV90] that with asymptotic probability 1 this sentence is equivalent to the above $\Sigma_1^1(\forall^3\exists)$ sentence with equality that expresses PARITY, so neither sentence has an asymptotic probability. Thus, the 0-1 law fails for the class $\Sigma_1^1(\forall^3\exists)$ without equality.

A similar argument applies to the class $\Sigma_1^1(\forall\exists\forall)$. Consider the sentence

$$(\exists U)(\exists S)\forall x\exists y\forall z[(E(x, z) \leftrightarrow S(y, z)) \wedge [(E(y, z) \leftrightarrow S(x, z)) \wedge (U(x) \leftrightarrow \neg U(y))].$$

It is shown in [Ved97] that this sentence expresses PARITY almost surely, so it has no asymptotic probability. Thus, the 0-1 law fails also for the class $\Sigma_1^1(\forall\exists\forall)$ without equality. (See also [Ten94].)

So far, the classifications of prefix classes according to their docility and according to the 0-1 law for the corresponding Σ_1^1 fragments seem to agree also for prefix classes without equality. Here also the difficult case was the Gödel class. Recall that the Gödel class without equality is docile. Nevertheless, Le Bars showed that the 0-1 law for the class $\Sigma_1^1(\text{Gödel})$ without equality fails [LeB98], confirming a conjecture in [KV90]. This implies that the two classification do not coincide without the presence of equality.

The failure of the 0-1 law for $\Sigma_1^1(\forall\forall\exists)$ without equality is demonstrated by showing that this class can express a certain property that does not have an asymptotic probability. Recall that a set U of nodes of a directed graph $G = (V, E)$ is *independent* if there are no edges between nodes in U and *dominating* if there is an edge from each node in $V - U$ to some node of U . We say that U is a *kernel* if it is both independent and dominating. The KERNEL property says that the graph has at least one kernel. It is easy to express KERNEL in $\Sigma_1^1(\forall\forall\exists)$ without equality:

$$(\exists U)(\forall x)(\forall y)(\exists z)[((U(x) \wedge U(y)) \rightarrow \neg E(x, y)) \wedge (\neg U(x) \rightarrow (U(z) \wedge E(x, z)))].$$

The KERNEL property has asymptotic probability 1 [dlV90]. Le Bars [LeB98] defined a variant of KERNEL, using a vocabulary with 16 binary relation symbols, that is also expressible in $\Sigma_1^1(\forall\forall\exists)$ without equality. He then showed that this property does not have an asymptotic probability.

Why do the docility classification and the 0-1 classification differ on the Gödel class? As has been already established in [Gol84], the “well-behavedness” of the Gödel class is very fragile. While the Gödel class without equality is docile, the class with equality is not. Thus, the addition of one built-in relation suffices to destroy the well-behavedness of the class. In the context of the 0-1 law, we are effectively adding a built-in relation—the random graph. Apparently, adding the random graph as a built-in relation also suffices to destroy the well-behavedness of the Gödel class.

While there is some intrinsic connection between docility and 0-1 laws (see discussion in [KV89]), the failure of the 0-1 law for the class $\Sigma_1^1(\forall\forall\exists)$ without equality shows that the two classifications need not be identical. In fact, Le Bars’s result demonstrates another instance of such a divergence. Consider fragments of first-order logic defined according to the number of individual variables used. Thus, FO^k is the set of first-order sentences with at most k variables. The unsolvability of the prefix class $\forall\exists\forall$ shows that FO^3 is unsolvable. On the other hand, it is known that FO^2 is solvable [Mor75] (in fact, satisfiability of FO^2 is NEXPTIME-complete [GKV97]). The failure of the 0-1 law for the class $\Sigma_1^1(\forall\exists\forall)$ implies its failure for the class $\Sigma_1^1(\text{FO}^3)$. Le Bars, however, showed that his variant of KERNEL can be expressed in $\Sigma_1^1(\text{FO}^2)$ without equality. Thus, FO^2 is docile even with equality, but $\Sigma_1^1(\text{FO}^2)$ without equality does not have a 0-1 law.

References

- [AHV95] Abiteboul, S, Hull, R., Vianu, V.: *Foundations of Databases*. Addison-Wesley, 1995.
- [AH78] Abramson, F.D., Harrington, L.A.: Models without indiscernibles. *J. Symbolic Logic* 43(1978), pp. 572–600.
- [BGG97] Börger, E., Grädel, E., Gurevich, Y.: *The Classical Decision Problem*. Springer-Verlag, 1997.

- [Bol85] Bollobas, B: *Random Graphs*. Academic Press, 1985
- [Che52] Chernoff, H.: A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observation. *Ann. Math. Stat.* 23(1952), pp. 493–509.
- [CKS81] Chandra, A., Kozen, D., Stockmeyer, L.: Alternation. *J. ACM* 28(1981), pp. 114–133.
- [Com88] Compton, K.J.: 0-1 laws in logic and combinatorics, in *NATO Adv. Study Inst. on Algorithms and Order* (I. Rival, ed.), D. Reidel, 1988, pp. 353–383.
- [dlV90] de la Vega, W.F.: Kernel on random graphs. *Discrete Mathematics* 82(1990), pp. 213–217.
- [DG79] Dreben, D., and Goldfarb, W.D.: *The Decision Problem: Solvable Classes of Quantificational Formulas*. Addison-Wesley, 1979.
- [EF95] Ebbinghaus, H.D., Flum, J.: *Finite Model Theory*, Springer-Verlag, 1995.
- [Fag74] Fagin, R.: Generalized first-order spectra and polynomial time recognizable sets. *Complexity of Computations* (R. Karp, ed.), SIAM–AMS Proc. 7(1974), pp. 43–73.
- [Fag76] Fagin, R.: Probabilities on finite models. *J. Symbolic Logic* 41(1976), pp. 50–58.
- [Gai64] Gaifman, H.: Concerning measures in first-order calculi. *Israel J. Math.* 2(1964), pp. 1–18.
- [GKLT69] Glebskii, Y.V., Kogan, D.I., Liogonkii, M.I., Talanov, V.A.: Range and degree of realizability of formulas in the restricted predicate calculus. *Cybernetics* 5(1969), pp. 142–154.
- [God32] Gödel, K.: Ein Spezialfall des Entscheidungsproblems der theoretischen Logik, *Ergebn. math. Kolloq.* 2(1932), pp. 27–28.
- [Gol84] Goldfarb, W.D.: The Gödel class with equality is unsolvable. *Bull. Amer. Math. Soc. (New Series)* 10(1984), pp. 113–115.
- [Gra83] Grandjean, E.: Complexity of the first-order theory of almost all structures. *Information and Control* 52(1983), pp. 180–204.
- [GKV97] Grädel, E., Kolaitis P.G., Vardi M.Y.: On the decision problem for two-variable first-order logic. *Bulletin of Symbolic Logic* 3(1997), 53–69.
- [GS83] Gurevich, Y., and Shelah, S.: Random models and the Gödel case of the decision problem. *J. of Symbolic Logic* 48(1983), pp. 1120–1124.
- [HIS85] Hartmanis, J., Immerman, N., Sewelson, J.: Sparse sets in NP-P – EXPTIME vs. NEXPTIME. *Information and Control* 65(1985), pp. 159–181.
- [Imm98] Immerman, N.: *Descriptive Complexity*. Springer-Verlag, 1998.
- [Kau87] Kaufmann, M.: *A counterexample to the 0-1 law for existential monadic second-order logic*. CLI Internal Note 32, Computational Logic Inc., Dec. 1987.
- [KS85] Kauffman, M., Shelah, S: On random models of finite power and monadic logic. *Discrete Mathematics* 54(1985), pp. 285–293.
- [KV87] Kolaitis, P., Vardi, M.Y.: The decision problem for the probabilities of higher-order properties. *Proc. 19th ACM Symp. on Theory of Computing*, New York, May 1987, pp. 425–435.
- [KV89] Kolaitis, P.G., Vardi, M.Y.: 0–1 laws for fragments of second-order logic - an overview. *Logic from Computer Science* (Proc. of Workshop, 1989), 1992, pp. 265–286.
- [KV90] Kolaitis, P., Vardi, M.Y.: 0-1 laws and decision problems for fragments of second-order logic. *Information and Computation* 87(1990), pp. 302–338.
- [Lac96] Lacoste: Finitistic proofs of 0-1 laws for fragments of second-order logic. *Information Processing Letters* 58(1996), pp. 1–4.
- [Lac97] Lacoste, T.: 0-1 Laws by preservation. *Theoretical Computer Science* 184(1997), pp. 237–245.
- [LeB98] Le Bars, J.M.: Fragments of existential second-order logic without 0-1 laws. *Proc. 13th IEEE Symp. on Logic in Computer Science*, 1998, pp. 525–536.

- [LeB00] Le Bars, J.M.: Counterexamples of the 0-1 law for fragments of existential second-order logic: an overview. *Bulletin of Symbolic Logic* 6(2000), pp. 67–82.
- [Mor75] Mortimer, M.: On language with two variables, *Zeit. für Math. Logik und Grund. der Math.* 21(1975), pp. 135–140.
- [NR77] Nešetřil, J., Rödl, V.: Partitions of finite relational and set systems. *J. Combinatorial Theory A* 22(1977), pp. 289–312.
- [NR83] Nešetřil, J., Rödl, V.: Ramsey classes of set systems. *J. Combinatorial Theory A* 34(1983), pp. 183–201.
- [Pos76] Pósa, L.: Hamiltonian circuits in random graphs. *Discrete Math.* 14(1976), pp. 359–364.
- [PS91] Pacholski, L., Szewast, L.: Asymptotic probabilities of existential second-order Gödel sentences. *Journal of Symbolic Logic* 56(1991), pp. 427–438.
- [PS93] Pacholski, L., Szewast, W.: A counterexample to the 0-1 law for the class of existential second-order minimal Gödel sentences with equality. *Information and Computation* 107(1993), pp. 91–103.
- [Ram28] Ramsey, F.P.: On a problem in formal logic. *Proc. London Math. Soc.* 30(1928), pp. 264–286.
- [Tra50] Trakhtenbrot, B.A.: The impossibility of an algorithm for the decision problem for finite models. *Doklady Akademii Nauk SSR* 70(1950), pp. 569–572.
- [Ten94] Tendera, L.: A note on asymptotic probabilities of existential second-order minimal classes: the last step. *Fundamenta Informaticae* 20(1994), pp. 277–285.
- [Ved97] Veda, A.: Asymptotic probabilities for second-order existential Kahr-Moore-Wang sentences. *J. Symbolic Logic* 62(1997), pp. 304–319.

On Algorithms and Interaction^{*}

Jan van Leeuwen¹ and Jiří Wiedermann²

¹ Department of Computer Science, Utrecht University,
Padualaan 14, 3584 CH Utrecht, the Netherlands.

² Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic.

Abstract. Many IT-systems behave very differently from classical machine models: they interact with an unpredictable environment, they never terminate, and their behavior changes over time. Wegner [25,26] (see also [28]) recently argued that the power of interaction goes beyond the Church-Turing thesis. To explore interaction from a computational viewpoint, we describe a generic model of an ‘interactive machine’ which interacts with the environment using single streams of input and output signals over a simple alphabet. The model uses ingredients from the theory of ω -automata. Viewing the interactive machines as transducers of infinite streams of signals, we show that their interactive recognition and generation capabilities are identical. It is also shown that, in the given model, all interactively computable functions are limit-continuous.

1 Introduction

Many modern systems in information technology have features that are no longer adequately described by the models of computation as we know them: their input is unpredictable and is not specified in advance, they do not terminate (unless a fault occurs), and they ‘learn’ over time. The systems are usually not meant to compute some final result, but are designed to interact with the environment and to develop or maintain some well-defined action-reaction behavior. Systems of this kind have previously been called ‘reactive systems’ and received much attention in the theory of concurrent processes (see e.g. [10,11] and [9,12]). Reactive systems are infinite-state systems and have a program and a behavior that is evolving over time.

Wegner [25,26] (see also Wegner and Goldin [28]) recently called for a more computational view of reactive systems, claiming that they have a richer behavior than ‘algorithms’ as we know them. He even challenged the validity of Church’s Thesis by proclaiming that Turing machines cannot adequately model the interactive computing behavior of typical reactive systems in practice. Wegner [26] (p. 318) writes:

^{*} This research was partially supported by GA ČR grant No. 201/98/0717 and by EC Contract IST-1999-14186 (Project ALCOM-FT).

“The intuition that computing corresponds to formal computability by Turing machines ... breaks down when the notion of what is computable is broadened to include interaction. Though Church’s thesis is valid in the narrow sense that Turing machines express the behavior of algorithms, the broader assertion that algorithms precisely capture what can be computed is invalid.”

For a discussion of this claim from a theoretical viewpoint, see Prasse and Rittgen [14]). The formal aspects of Wegner’s theory of interaction are studied in e.g. Wegner and Goldin [27,29].

Irrespective of whether this claim is valid or not, it is interesting to look at the implications of reactivity, or interactivity, from a computational viewpoint. In [19] we have explored this for embedded systems, in [20] for ‘dynamic’ networks like the internet. The cornerstone in these studies is the notion of an interactive machine. In the present paper we describe a possible model of an interactive machine C , interacting with an environment E using single streams of input and output signals over a simple alphabet. Our notion of interactive machine (IM) is very similar to Broy’s notion of component ([1]), but we restrict ourselves to deterministic components only. We will point to several connections to the well-known theory of ω -automata, and there clearly are game-theoretic aspects as well.

We will identify a special condition, referred to as the interactivity condition, which we will impose throughout. Loosely speaking, the condition states that C is guaranteed to give some meaningful output within finite time any time after receiving a meaningful input from E and vice versa. We assume that C is a ‘learning’ program with unbounded memory, with a memory contents that is building up over time and that is never erased unless the component explicitly does so. This compares to the use of persistent Turing machines by Goldin [4] (see also Goldin and Wegner [5]) and Kosub [8]. No special assumptions are made about the ‘speed’ at which C and E can operate or generate responses. We assume that E can behave arbitrarily and unpredictably.

Viewing IM’s as interactive transducers of the signals that they receive from the environment we show that, using suitable definitions, recognition and generation coincide just like they do in the case of Turing machines. The proof depends on some of the special operational assumptions in the model. We also define a general notion of interactively computable functions. We prove that interactively computable functions are limit-continuous, using a suitable extension of the notion of continuity known from the semantics of computable functions. It is shown that interactively computable 1-1 functions have interactively computable inverses.

The study of ‘machines’ working on infinite input streams (ω -words) is by no means new and has a sizable literature, with the first studies dating back to the nineteen sixties and seventies (cf. [15], [17]). In the model studied in the present paper a number of features are added that are meant to better capture some intuitive notions of interactivity, inspired by Wegner’s papers. The present paper derives from [19].

2 A Simple Model of Interactive Computation

Let C be a component (a software agent or a device) that interacts with an environment E . We assume that C and E interact by exchanging signals (symbols). Although general interactive systems do not need to have a limit on the nature and the size of the signals that they exchange, we assume that the signals are taken from a fixed and finite alphabet. More precisely:

(*Alphabet*) C and E interact by exchanging symbols from the alphabet $\Sigma = \{0, 1, \tau, \#\}$.

Here 0 and 1 are the ordinary bits, τ is the ‘silent’ or empty symbol, and $\#$ is the fault or error symbol. Instead of the bits 0 and 1, one could use any larger (finite) set of symbols but this is easily coded back into binary form. We assume that the interaction is triggered by E .

In order to describe the interactions between C and E we assume a uniform time-scale of discrete moments $0, 1, 2, \dots$. C and E are assumed to work synchronously. At any time t , E can send a symbol of Σ to C and C can send a symbol of Σ to E . It is possible that E or C remain ‘silent’ for a certain amount of time, i.e. that either of them does not send any real signal during some consecutive time moments. During these moments E or C is assumed to send the symbol τ just to ‘record’ this. For the symbol $\#$ a special convention is used:

(*Fault rule*) If C receives a symbol $\#$ from E , then it will output a $\#$ in finite time after that as well (and vice versa).

If no $\#$ ’s are exchanged, the interaction between E and C is called fault-free (error-free). The communication between E and C can be described by the sequences $e = e_0 e_1 \dots e_t \dots$ and $c = c_0 c_1 \dots c_t \dots$, with e_t denoting the signal that E sends to C at time t and c_t denoting the signal that C sends to E at time t . It is assumed that E and C send at most one signal per time-moment and that a signal sent at time t is received by the other party at the same time-moment. When E or C is silent at time t , then the corresponding symbol sent is τ (‘empty’). If two sequences correspond to the actual interactions that take place over time, we say that they form an ‘interaction pair’ and represent an *interactive computation* of C in response to the (unpredictable) environment E . We let $\bar{e}$ and $\bar{c}$ denote the sequences e and c *without* the τ ’s.

We assume the following property as being characteristic for interactive computations: E sends signals to C infinitely often and C is guaranteed to give an output within finite time, any time after receiving an input signal from the E . More precisely:

(*Interactiveness*) For all times t , when E sends a non-empty signal to C at time t , then C sends a non-empty signal to E at some time t' with $t' > t$ (and vice versa).

The condition of interactiveness is assumed throughout this paper. Note that, in the definition of interactiveness, we do not make any special assumption about the causality between the signal sent (by E or C) at time t and the signal sent (by C or E respectively) at time t' . With the condition of interactiveness into effect, the behavior of a component C with respect to E is a relation on infinite strings over Σ , accounting for the intermittent empty symbols in the proper way (i.e. in the interaction they are present but in the reported sequences they are suppressed). We assume that E sends a non-empty signal at least once, thus always triggering an infinite interaction sequence.

We assume that C has only one channel of interaction with E , admits multi-threading (allowing it to run several processes simultaneously), and has a fixed but otherwise arbitrary speed of operation. As a consequence it is possible for C to have a foreground process doing e.g. the I/O-operations very quickly and have one or more background processes running at a slower pace. We also assume:

(Unbounded component memory) C works without any a priori bounds on the amount of available memory, i.e. its memory space is potentially infinite.

In particular C 's memory is never reset during an interactive computation, unless its program explicitly does so. We allow C to build up an 'infinite' database of knowledge that it can consult in the course of its interactions with the environment.

The environment E can behave like an adversary and can send signals in any arbitrary, unpredictable and 'non-algorithmic' way. We do assume it has the full choice of symbols from Σ at its disposal at all times. The fact that E can react in many different ways at any given moment accounts for the variety of output sequences C can produce (generate), when we consider all possible behaviors later on. We assume that, in principle, E can generate any possible signal at any moment but do not concern ourselves with the way E actually comes up with a specific signal in any given circumstance.

Despite the assumed generality, it is conceivable that E is constraint in some way and can generate at some or any given moment only a *selection* of the possible signals from Σ . The constraints could serve to define partial computations, defined only for subsets of possible inputs. If this is the case, we assume that the constraints are of an algorithmic nature (i.e. it can be checked algorithmically that E 's response is within the allowed constraints) and never limit the infinity of the interaction. A component C that interacts with its environment according to the given assumptions, will be called an interactive machine (IM). We assume its operation is governed by an evolving process that can be effectively simulated.

It will be helpful to describe an interactive computation of C and E also as a mapping (transduction) of strings e (of environment input) to strings c (of component responses). In this way C acts as an ω -transducer on infinite strings, with the special incremental and interactive behavior as described here (and with game-theoretic connotations). Considerable care is needed in dealing with

sequences of τ 's. A sequence of 'silent steps' can often be meaningful to C or E , e.g. for internal computation or timing purposes. It means that a given infinite string $\bar{e}$ may have a multitude of different transductions c , depending on the way E cares to intersperse $\bar{e}$ with 'empty' signals. One should note that E is fully unpredictable in this respect. However, the assumed interactivensness forces E to keep sequences of intermittent empty signals finite.

For the purposes of this paper we assume that the environment E sends a non-empty signal at *every* moment in time. We retain the possibility for C to emit τ 's and thus to do internal computations for some time without giving output, even though the assumed interactivensness will force C to give some non-empty output after finite time also.

Definition 1. *The behavior of C with respect to E is the set $T_C = \{(e, \bar{c}) | (e, c) \text{ is an interaction pair of } C \text{ and } E\}$. If (e, c) is an interaction pair of C and E , then we also write $T_C(e) = \bar{c}$ and say that $\bar{c}$ is the interactive transduction of e by C .*

Definition 2. *A relation T on infinite strings is called interactively computable if there is an interactive machine C such that $T = T_C$.*

Seemingly simple transductions may be impossible in interactive computation, due to the strict requirement of interactivensness and the unpredictability of the environment. Let 0^* denote the set of finite sequences of 0's (including the empty sequence), 2^* the set of all finite strings over $\{0, 1\}$, and 2^ω the set of infinite strings over $\{0, 1\}$. It can easily be argued that no interactive machine C can exist that transduces its inputs such that input streams that happen to be of the form $1\alpha 1\beta 1\gamma$ are transduced to $1\beta 1\alpha 1\gamma$, with $\alpha, \beta \in 0^*$ and $\gamma \in 2^\omega$. The example crucially depends on the fact that C cannot predict that the input will be of the special form specified for the transduction.

3 Interactive Relations and Programs

Given a sequence $u \in 2^\omega$ and $t \geq 0$, let $\text{pref}_t(u)$ be the length- t prefix of u . For finite or infinite strings u we write $x \prec u$ if x is a finite (and strict) prefix of u . If x is a finite string, denote its length by $|x|$. Modify the common definition of monotonic functions (cf. [30]) for the case of partial functions as follows.

Definition 3. *A partial function $g : 2^* \rightarrow 2^*$ is called monotonic if for all $x, y \in 2^*$, if $x \prec y$ and $g(y)$ is defined then $g(x)$ is defined as well and $g(x) \preceq g(y)$.*

The following observation captures some aspects of the computational model in terms of monotonic functions.

Theorem 1. *If a relation $T \subseteq 2^\omega \times 2^\omega$ is interactively computable, then there exists a classically computable, monotonic partial function $g : 2^* \rightarrow 2^*$ such that for all $(u, v) \in 2^\omega \times 2^\omega$, $(u, v) \in T$ if and only if for all $t \geq 0$ $g(\text{pref}_t(u))$ is defined, $\lim_{t \rightarrow \infty} |g(\text{pref}_t(u))| = \infty$ and for all $t \geq 0$ $g(\text{pref}_t(u)) \prec v$.*

Proof. Let $T = T_C$. We define g by designing a (Turing) machine M_g for it. Given an arbitrary finite string $x = x_0x_1 \dots x_{t-1} \in 2^*$ on its input tape, M_g operates as follows. M_g simulates C using the ‘program’ of C , feeding it the consecutive symbols of x as input and checking every time it does so whether the next symbol is an input signal that E *could* have given on the basis of the interaction with C up until this moment. To check this, M_g employs the verifier $\mathcal{E}$ which checks the possible constraints on E (and which evolves along with the simulation). As long as no inconsistency is detected, M_g continues with the simulation of the interaction of E and C . Whenever the simulation leads C to output a signal 0 or 1, M_g writes the corresponding symbol to its output tape. When the simulation leads C to output a τ , M_g writes nothing. When the simulation leads C to output a $\sharp$ or when the verifier detects that the input is not consistent with E ’s possible behavior, then M_g is sent into an indefinite loop. If machine M_g has successfully completed the simulation up to and including the simulation of the (processing of the) final input symbol x_{t-1} , then M_g halts. It follows that M_g terminates if and only if x is a valid beginning of an interaction of E with C , with C ’s response appearing on the output tape if it halts. The result now follows by observing what properties of g are needed to capture that $(u, v) \in T$ in terms of M_g ’s action on the prefixes of u . The constraints capture the interactiveness of C and E and the fact that the interaction must be infinite. It is clear from the construction that g is monotonic.

For at least one type of interactively computable relation can the given observation be turned into a complete characterisation. Let a relation $T \subseteq 2^\omega \times 2^\omega$ be called *total* if for every $u \in 2^\omega$ there exists a $v \in 2^\omega$ such that $(u, v) \in T$. Note that the behaviors of interactive machines without special input constraints are always total relations. In the following result the monotonicity of g is *not* assumed beforehand.

Theorem 2. *Let $T \subseteq 2^\omega \times 2^\omega$ be a total relation. T is interactively computable if and only if there exists a classically computable total function $g : 2^* \rightarrow 2^*$ such that for all $(u, v) \in 2^\omega \times 2^\omega$, $(u, v) \in T$ if and only if $\lim_{t \rightarrow \infty} |g(\text{pref}_t(u))| = \infty$ and for all $t \geq 0$ $g(\text{pref}_t(u)) \prec v$.*

Proof. The ‘only if’ part follows by inspecting the proof of theorem 1 again. If T is total, then the constructed function g is seen to be total as well and T and g satisfy the stated conditions.

For the ‘if’ part, assume that $T \subseteq 2^\omega \times 2^\omega$ is a total relation, that g is a computable total function and that for all $(u, v) \in 2^\omega \times 2^\omega$, $(u, v) \in T$ if and only if $\lim_{t \rightarrow \infty} |g(\text{pref}_t(u))| = \infty$ and for all integers $t \geq 0$ $g(\text{pref}_t(u)) \prec v$. To prove that T is interactively computable, design an interactive machine C that operates as follows.

While E feeds input, a foreground process of C keeps buffering the input symbols in a queue $q = q_0q_1 \dots q_t$ with $t \rightarrow \infty$. Let $r \in 2^*$ be the (finite) output generated by C at any given moment. We will maintain the invariant that q is a prefix of u and r a prefix of v for some pair $(u, v) \in T$ (which exists by the totality of T). Letting q grow into ‘ u ’ by the input from E , we let r grow into

‘ v ’ by letting C carry out the following background process P every once in a while. C uses a counter c_q that is initially set to 1. C outputs ‘empty’ signals as long as a call to P is running.

When called, P copies the length- c_q prefix of the queue into x , it increments c_q by one, and computes $g(x)$ using the subroutine for g . (Note that the string now in x extends the string on which the previous call of P operated by precisely one symbol.) By totality of g the subroutine ends in finitely many steps and yields a string y as output. By totality of T and the second condition on g only two cases can occur: $r \prec y$ or $y \preceq r$. If $r \prec y$, then C outputs the symbols by which y extends r one after the other (implicitly updating r as well to account for the new output) and calls P again after it has done so. If $y \preceq r$, C does not generate any special output and simply moves on to another call of P provided at least one further input symbol has entered the queue in the meantime. Because $\lim_{t \rightarrow \infty} |g(\text{pref}_t(u))| = \infty$, there will be infinitely many calls to P in which the case ‘ $r \prec y$ ’ occurs. Thus r will grow to infinity, with $\lim_{t \rightarrow \infty} r = v$ precisely being the output generated by C .

Combining the two theorems we conclude: *the interactively computable total relations are precisely the relations implied ‘in the limit’ by classically computable, monotonic total functions on 2^** . We will return to this characterisation of interactive computations in Section 6.

It is quite realistic to assume that the initial specification of C is a program, written in some acceptable programming system. For example, the internal operation of C could be modeled by a persistent Turing machine of some sort (as in Wegner and Goldin [27] and Goldin [4]). It is easily argued that interactivity, as a property of arbitrary programs, is recursively undecidable. The following stronger but still elementary statement can be proved, thanking discussions with B. Rovan: *the set of interactive programs is not recursively enumerable* ([19]).

In the further sections we will study the given model of interactive computing from three different perspectives: recognition, generation and translation (i.e. the computation of mappings on infinite binary strings).

4 Interactive Recognition

Interactive machines typically perform a task in *monitoring*, i.e. the online perception or recognition of patterns in infinite streams of signals from the environment. The notion of recognition is well-known and well-studied in the theory of ω -automata (cf. [15], [16,17]) and is usually based on the automaton passing an infinite number of times through one or more ‘accepting’ states during the processing of the infinite input string. In interactive systems this is not detectable (cf. proposition 1 (v) below) and thus not applicable. A component is normally placed in an environment that follows a certain specification, and the component has to observe that this is adhered to. This motivates the following definition.

Definition 4. *An infinite string $\alpha \in 2^\omega$ is said to be recognized by C if $(\alpha, 1^\omega) \in T_C$.*

The definition says that in our model an infinite string α is recognized if C outputs a 1 every once in a while and *no* other symbols except τ 's in between during the interactive computation with E , where E generates the infinite string α as input during the computation.

Definition 5. *The set interactively recognized by C with respect to E is the set $J_C = \{\alpha \in 2^\omega \mid \alpha \text{ is recognized by } C\}$.*

Note that interactiveness is required on all infinite inputs, i.e. also on the strings that are not recognized. As a curious fact we observe that when it comes to recognition we may assume that C makes no silent steps: letting C output a 1 whenever it wants to output a τ does not affect the set of strings recognized by the component.

Definition 6. *A set $J \subseteq 2^\omega$ is called interactively recognizable if there exists a component C such that $J = J_C$.*

Proposition 1. *The following set is interactively recognizable:*

- (i) $J = \{\alpha \in 2^\omega \mid \alpha \text{ contains at most } k \text{ ones}\}$, for any fixed integer k ,
but the following sets are not interactively recognizable:
- (ii) $J = \{\alpha \in 2^\omega \mid \alpha \text{ contains precisely } k \text{ ones}\}$, for any fixed integer $k \geq 1$,
- (iii) $J = \{\alpha \in 2^\omega \mid \alpha \text{ contains finitely many } 1\text{'s}\}$,
- (iv) $J = \{\alpha \in 2^\omega \mid \alpha \text{ contains at least } k \text{ ones}\}$, for any fixed integer $k \geq 1$,
- (v) $J = \{\alpha \in 2^\omega \mid \alpha \text{ contains infinitely many } 1\text{'s}\}$.

Proof. We illustrate the typical argument for (iii). Suppose there was an interactive component C that recognized J . Let E input 1's. By interactiveness C must generate a non-empty signal σ at some moment in time. E can now fool C as follows. If $\sigma = 0$, then let E switch to inputting 0's from this moment onward: it means that the resulting input belongs to J but C does not respond with all 1's. If $\sigma = 1$, then let E continue to input 1's. Possibly C outputs a few more 1's but there must come a moment that it outputs a 0. If it didn't then C would recognize the string $1^\omega \notin J$. As soon as C outputs a 0, let E switch to inputting 0's from this moment onward: it means that the resulting input still belongs to J but C does not recognize it properly. Contradiction.

The different power of interactive recognition is perhaps best expressed by the following observation. We assume again that the internal operation of the components that we consider is specified by some program in an acceptable programming system. For sets $J \subseteq 2^\omega$, let $\text{Init}(J)$ be the set of all finite prefixes of strings from J . Similar to a result of Cohen and Gold ([2], theorem 7.22) one can show that the initial part of an environment input need not hold any clue about the recognizability of the whole string, as one would expect.

Corollary 1. *There are interactively recognizable sets J such that $\text{Init}(J)$ is not recursively enumerable.*

Proof. Consider the set $J = \{0^n 1 \{0, 1\}^\omega | n \in A\} \cup 0^\omega$ for a non-recursively enumerable set A whose complement is recursively enumerable. It can be shown that J is interactively recognizable. If $\text{Init}(J)$ were recursively enumerable, then so would $\text{Init}(J) \cap 0^* 1 = \{0^n 1 | n \in A\}$ and thus A . Contradiction.

Finally we note some rules for constructing new interactively recognizable sets from old ones. The proofs are slightly tedious because of the mechanism of feedback that may exist from the output of a component back to E .

Proposition 2. *The family of interactively recognizable sets of infinite strings is*

- (i) *closed under $\cup$ and $\cap$, but*
- (ii) *not closed under ω -complement (i.e. complement in 2^ω).*

Proof. For (i) we refer to [19]. For (ii), consider the set $J = 0^\omega \cup 0^* 1 0^\omega$. By proposition 1 (i) the set is interactively recognizable, but by proposition 1 (iv) its ω -complement is not.

5 Interactive Generation

Interactive machines typically also perform tasks in *controlling* other components. This involves the online translation of infinite streams into other, more specific streams of signals. In this section we will consider what infinite streams of signals an interactive machine can *generate*. The notion of generation is well-known in automata theory and related to matters of definability and expressibility, but it seems not to have been looked at very much in the theory of ω -automata (cf. [15]). Our aim will be to prove that generation and recognition are *duals* in our model. The following definition is intuitive.

Definition 7. *An infinite string $\beta \in 2^\omega$ is said to be generated by C if there exists an environment input $\alpha \in \{0, 1\}^\omega$ such that $(\alpha, \beta) \in T_C$.*

Unlike the case for recognition (cf. section 3) one cannot simplify the output capabilities for components C now. In particular we have to allow that C outputs $\sharp$ -symbols when it wants to signify that its generation process has gotten off on some wrong track. If C outputs a $\sharp$ -symbol, it will automatically trigger E to produce a $\sharp$ as well some finite time later. Strings that contain a $\sharp$ are *not* considered to be validly generated.

Definition 8. *The set interactively generated by interactive machine C is the set $L_C = \{\beta \in 2^\omega | \beta \text{ is generated by } C\}$.*

Observe that, contrary to recognition, C may need to make silent steps while generating. It means that interactive generation is not necessarily an online process.

Definition 9. *A set $L \subseteq 2^\omega$ is called interactively generable if there exists a component C such that $L = L_C$.*

In the context of generating ω -strings, it is of interest to know what finite prefixes an interactive machine C can generate. To this end we consider the following intermediate problem:

(*Reachability*) Given an interactive machine C and a finite string $\gamma \in \{0, 1\}^*$, is there an interactive computation of C such that the sequence of non-silent symbols generated and output by C at some finite moment equals γ .

It can be shown that the reachability problem for interactive machines C is effectively decidable. The proof involves an application of *König's Unendlichkeitslemma* ([6,7]). We will now show that the fundamental law that 'what can be generated can be recognized and vice versa' holds for interactive machines. We prove it in two steps.

Lemma 1. *For all sets $J \subseteq 2^\omega$, if J is interactively generable then J is interactively recognizable.*

Proof. Let J be interactively generated by some interactive machine C , i.e. $J = L_C$. To show that J can be interactively recognized, design the following machine C' . Let the input from E be e . C' buffers the input that it receives from E symbol after symbol, and goes through the following cycle of activity: it takes the string r that is currently in the buffer, decides whether r is reachable for C (applying *Reachability*), and outputs a 1 if it is and a 0 if it is not. This cycle is repeated forever, each time taking the new (and longer) string r that is in the buffer whenever a new cycle starts again. Because the reachability problem is decidable in finite time, C' is an interactive machine. Clearly, if an ω -string belongs to J then all its prefixes are reachable for C and C' recognizes it. Conversely, if an ω -string e is recognized by C' then it must be interactively generated by C and hence belong to J . The latter can be argued using the *Unendlichkeitslemma* again.

Lemma 2. *For all sets $J \subseteq 2^\omega$, if J is interactively recognizable then J is interactively generable.*

Proof. Let J be interactively recognizable. Let C be an interactive machine such that $J = L_C$. To show that J can be interactively generated, design the following machine C' . C' buffers the input that it receives from E symbol after symbol, and copies it to output as well (at a slower pace perhaps). At the same time C' runs a simulation of C in the background, inputting the symbols from the buffer one after the other and reacting like E would, keeping track whether the buffered input is exactly the sequence that E would input when taking C 's responses into account. Let C' continue to copy input to output as long as no inconsistency between the buffered string and E 's simulated behavior arises and as long as C outputs τ 's and 1's (in the simulation). If anything else occurs, let C' switch to outputting $\sharp$'s. C' is clearly interactive, and the generated strings are precisely those that C recognizes.

From the given lemmas we conclude the following basic result.

Theorem 3. *For all sets $J \subseteq 2^\omega$, J is interactively generable if and only if J is interactively recognizable.*

The theorem gives evidence that the concepts of interactive recognition and generation as intuitively defined, are well-chosen. Besides being of interest in its own right, theorem 3 also enables one to prove that certain sets are not interactively recognizable by techniques different from Section 4.

6 Interactive Translations

As noted in the previous section, interactive machines typically perform the online translation of infinite streams into other infinite streams of signals. We will consider this further, viewing interactive machines as transducers and viewing the translations (or: transductions) they realize as interactive mappings defined on infinite strings of 0's and 1's. The related notion of ω -transduction in the theory of ω -automata has received quite some attention before (cf. Staiger [15]), but mostly in the context of ω -regular languages only. In this section we will present some basic observations on interactive mappings. Let C be an interactive machine. Let T_C be the behavior of C (cf. definition 2).

Definition 10. *The interactive mapping computed by C is the mapping $f_C : 2^\omega \rightarrow 2^\omega$ defined by the following property: $f_C(\alpha) = \beta$ if and only if $(\alpha, \beta) \in T_C$.*

Generally speaking, an interactive mapping is a partial function defined over infinite binary strings. If $f_C(\alpha) = \beta$ (defined), then C actually outputs a sequence $s \in \{0, 1, \tau\}^\omega$ in response to input $\alpha \in \{0, 1\}^\omega$ such that $\bar{s} = \beta$.

Definition 11. *A partial function $f : 2^\omega \rightarrow 2^\omega$ is called interactively computable if there exists an interactive machine C such that $f = f_C$.*

Computable functions on incomplete inputs (like finite segments of infinite strings) should be continuous in the sense that, at any moment, any further extension of the input should lead to an extension of the output as it is at that moment and vice versa, without retraction of any earlier output signals. Interactively computable functions clearly all have this property on defined values, which can be more precisely formulated as follows. Modify the classical definition of 'continuous functions' (cf. [30]) for the case of functions on infinite strings as follows.

Definition 12. *A partial function $f : 2^\omega \rightarrow 2^\omega$ is called limit-continuous if there exists a classically computable partial function $g : 2^\star \rightarrow 2^\star$ such that the following conditions are satisfied:*

- (1) *g is monotonic, and*
- (2) *for all strictly increasing chains $u_1 \prec u_2 \prec \dots \prec u_t \prec \dots$ with $u_t \in 2^\star$ for $t \geq 1$, one has $f(\lim_{t \rightarrow \infty} u_t) = \lim_{t \rightarrow \infty} g(u_t)$.*

In condition (2) the identity is assumed to hold as soon as one of the sides is defined. Clearly monotonic functions map chains into chains, if they are defined on all elements of a chain, but they do not necessarily map strictly increasing chains into chains that are strictly increasing again. Definition 13 implies that if a total function f is limit-continuous, then the corresponding g must be total as well and also map strictly increasing chains into ultimately increasing chains. Using theorems 1 and 2 we now easily conclude the following facts.

Theorem 4. *If $f : 2^\omega \rightarrow 2^\omega$ is interactively computable, then f is limit-continuous.*

Theorem 5. *Let $f : 2^\omega \rightarrow 2^\omega$ be a total function. Then f is interactively computable if and only if f is limit-continuous.*

Several other properties of interactively computable functions are of interest. For example, without assuming that either f or g is total, it can be shown that if f and g are interactively computable, then so is $f \circ g$ (cf. [19]). The following result is more tedious. It shows that the inverses of 1-1 mappings defined by interactive machines are interactively computable again.

Theorem 6. *Let f be interactively computable and 1-1. Then f^{-1} is interactively computable as well.*

Proof. Let $f = f_C$ and assume f is 1-1. If $f(\alpha) = \beta$ (defined) then $f^{-1}(\beta) = \alpha$, and we seek an interactive machine to realize this. Design a machine C' that works as follows.

Let the input supplied so far be r , a finite prefix of ' β '. Assume the environment supplies further symbols in its own way, giving C' a longer and longer part r of the string β to which an original under f is sought. Let C' buffer r internally. We want the output $\sigma = \bar{u}$ of C' at any point to be a finite (and growing) prefix of ' α '. Let this be the case at some point. Now let C' do the following, with more and more symbols coming in and 'revealing' more and more of β .

The dilemma C' faces is whether to output a 0 or a 1 (or, of course, a $\sharp$). In other words, C' must somehow decide whether $\sigma 0$ or $\sigma 1$ is the next longer prefix of the original α under f in the unfolding of β . We argue that this is indeed decidable in finite time. The idea is to look 'into the future' and see which of the two possibilities survives. To this end, we create two processes P_b ($b = 0, 1$) that explore the future for σb . A P_b works as follows. Remember that symbols continue to come into r and that C' outputs τ 's until it knows better.

P_b works on the infinite binary tree $\mathcal{T}$ with its left and right branches labeled 0 and 1, respectively. Every node q of $\mathcal{T}$ can be represented by a finite string p_q , consisting of the 0's and 1's on the path from the root down to q . P_b labels the root by 'Y'. Then it works through the unlabeled nodes q level by level down the tree, testing for every node q whether the string $\sigma b p_q$ is output (i.e. is reachable) by C as it operates on (a prefix of) r , i.e. on a prefix of β . (P_b does this in the usual way, by running the simulation of the interactive computation of C and E .) If $\sigma b p_q$ is reachable, then label q by 'Y'. If the output of C does not reach

the end of σbp_q but is consistent with it as far as it gets, then P_b waits at this node and continues the simulation as more and more symbols come into r . (By interactivity, r will eventually be long enough for C to give an output at least as long as σbp_q .) If the output of C begins to differ from σbp_q before the end of σbp_q is reached, then label q by ‘N’ and prune the tree below it. If the simulation runs into an inconsistency between E ’s behavior and the string r that is input, then label q by ‘N’ and prune the tree below it as well. If P_b reaches a tree level where all nodes have been pruned away, it stops. Denote the tree as it gets labeled by P_b by $\mathcal{T}_b$.

Let C' run P_0 and P_1 ‘in parallel’. We claim that one of the two processes must stop in finite time. Suppose that neither of the two stopped in finite time. Then $\mathcal{T}_0$ and $\mathcal{T}_1$ would both extend to infinite trees as r extends to ‘infinity’ (i.e. turning into the infinite string β). By the *Unendlichkeitslemma*, $\mathcal{T}_0$ will contain an infinite path δ_0 and likewise $\mathcal{T}_1$ will contain an infinite path δ_1 . This clearly implies that both $\sigma 0\delta_0$ and $\sigma 1\delta_1$ would be mapped by C to β , which contradicts that f is $1 - 1$. It follows that at least one of the processes P_0 and P_1 must stop in finite time. (Stated in another way, the process that explores the wrong prefix of α will die out in finite time.) Note that both processes could stop, which happens at some point in case the limit string β has *no* original under f .

Letting C' run P_0 and P_1 ‘in parallel’, do the following as soon as one of the processes stops. If both processes stop, C' outputs $\sharp$. If P_0 stopped but P_1 did not, then output 1. If P_1 stopped but P_0 did not, then output 0. If C' output a b (0 or 1), then it repeats the whole procedure with σ replaced by σb . If it output a $\sharp$ it will continue to output $\sharp$ ’s from this moment onwards. It is easily seen that C' is interactive and that it computes precisely the inverse of f .

7 Conclusion

In this paper we considered a simple model of interactive computing, consisting of a machine interacting with an ‘environment’ on infinite streams of input and output symbols that are exchanged in an online manner. The motivation stems from the renewed interest in the computation-theoretic capabilities of interactive computing, in particular by Wegner’s claim (cf. [25]) that ‘interactive computing is more powerful than algorithms’. In the model we have tried to identify a number of properties which one would intuitively ascribe to components of interactive system, to obtain a generic model of an interactive machine. In [19] and [20] we have used it to model some fundamental interactive features of embedded systems and of the internet, respectively. In the latter case, the model includes the possibility of letting external information enter into the interaction process. Such an intervention may take the form of e.g. a software or hardware upgrade of the underlying device in the course of its interaction with its environment. The resulting model presents a kind of a non-uniformly evolving, interactive device that possesses a computational power that outperforms that of the interaction machines considered in this paper. The steadily increasing role of interactive

computing and the understanding of its computational power has led us to propose an extension of the paradigm of ‘classical’, non-interactive Turing machine computing in favor of an interactive, possibly non-uniform machine model [21] that captures the nature of contemporary computing. Many interesting problems remain in the further understanding of interactive computing.

Acknowledgements. We are grateful to D.Q. Goldin and T. Muntean for several useful comments and suggestions on a preliminary version of [19].

References

1. M. Broy. A logical basis for modular software and systems engineering, in: B. Rovan (Ed.), *SOFSEM'98: Theory and Practice of Informatics*, Proc. 25th Conference on Current Trends, Lecture Notes in Computer Science, Vol. 1521, Springer-Verlag, Berlin, 1998, pp. 19-35.
2. R.S. Cohen, A.Y. Gold. ω -Computations on Turing machines, *Theor. Comput. Sci.* 6 (1978) 1-23.
3. J. Engelfriet, H.J. Hoogeboom. X -automata on ω -words, *Theor. Comput. Sci.* 110 (1993) 1-51.
4. D.Q. Goldin. Persistent Turing machines as a model of interactive computation, in: K-D. Schewe and B. Thalheim (Eds.), *Foundations of Information and Knowledge Systems*, Proc. First Int. Symposium (FoIKS 2000), Lecture Notes in Computer Science, vol. 1762, Springer-Verlag, Berlin, 2000, pp. 116-135.
5. D. Goldin, P. Wegner. Persistence as a form of interaction, Techn. Rep. CS-98-07, Dept. of Computer Science, Brown University, Providence, RI, 1998.
6. D. König. Sur les correspondances multivoques des ensembles, *Fundam. Math.* 8 (1926) 114-134.
7. D. König. Über eine Schlussweise aus dem Endlichen ins Unendliche (Punktmengen. – Kartenfärben. – Verwandtschaftsbeziehungen. – Schachspiel), *Acta Litt. Sci. (Sectio Sci. Math.)* 3 (1927) 121-130.
8. S. Kosub. Persistent computations, Technical Report No. 217, Institut für Informatik, Julius-Maximilians-Universität Würzburg, 1998.
9. Z. Manna, A. Pnueli. Models for reactivity, *Acta Informatica* 30 (1993) 609-678.
10. R. Milner. *A calculus of communicating systems*, Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, Berlin, 1980.
11. R. Milner. Elements of interaction, *C.ACM* 36:1 (1993) 78-89.
12. A. Pnueli. Applications of temporal logic to the specification and verification of reactive systems: a survey of current trends, in: J.W. de Bakker, W.-P. de Roever and G. Rozenberg, *Current Trends in Concurrency*, Lecture Notes in Computer Science, Vol. 224, Springer-Verlag, Berlin, 1986, pp. 510-585.
13. A. Pnueli. Specification and development of reactive systems, in: H.-J. Kugler (Ed.), *Information Processing 86*, Proceedings IFIP 10th World Computer Congress, Elsevier Science Publishers (North-Holland), Amsterdam, 1986, pp. 845-858.
14. M. Prasse, P. Rittgen. Why Church's Thesis still holds. Some notes on Peter Wegner's tracts on interaction and computability, *The Computer Journal* 41 (1998) 357-362.

15. L. Staiger. ω -Languages, in: G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 3: *Beyond Words*, Chapter 6, Springer-Verlag, Berlin, 1997, pp. 339-387.
16. W. Thomas. Automata on infinite objects, in: J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, Vol. B: *Models and Semantics*, Elsevier Science Publishers, Amsterdam, 1990, pp. 135-191.
17. W. Thomas. Languages, automata, and logic, in: G. Rozenberg and A. Salomaa (Eds.), *Handbook of Formal Languages*, Vol. 3: *Beyond Words*, Chapter 7, Springer-Verlag, Berlin, 1997, pp. 389-455.
18. B.A. Trakhtenbrot. Automata and their interaction: definitional suggestions, in: G. Ciobanu and G. Păun (Eds.), *Fundamentals of Computation Theory*, Proc. 12th International Symposium (FCT'99), Lecture Notes in Computer Science, Vol. 1684, Springer-Verlag, Berlin, 1999, pp. 54-89.
19. J. van Leeuwen, J. Wiedermann. A computational model of interaction, Technical Report, Dept of Computer Science, Utrecht University, Utrecht, 2000.
20. J. van Leeuwen, J. Wiedermann. Breaking the Turing barrier: the case of the Internet, Technical Report, Inst. of Computer Science, Academy of Sciences, Prague, 2000.
21. J. van Leeuwen, J. Wiedermann. Extending the Turing Machine Paradigm, in: B. Engquist and W. Schmidt (Eds.), *Mathematics Unlimited - 2001 and Beyond*, Springer-Verlag, Berlin, 2000 (to appear).
22. K. Wagner, L. Staiger. Recursive ω -languages, in: M. Karpinsky (Ed.), *Fundamentals of Computation Theory*, Proc. 1977 Int. FCT-Conference, Lecture Notes in Computer Science, Vol. 56, Springer-Verlag, Berlin, 1977, pp. 532-537.
23. P. Wegner. Interactive foundations of object-based programming, *IEEE Computer* 28:10 (1995) 70-72.
24. P. Wegner. Interaction as a basis for empirical computer science, *Comput. Surv.* 27 (1995) 45-48.
25. P. Wegner. Why interaction is more powerful than algorithms, *C.ACM* 40 (1997) 80-91.
26. P. Wegner. Interactive foundations of computing, *Theor. Comp. Sci.* 192 (1998) 315-351.
27. P. Wegner, D. Goldin. Co-inductive models of finite computing agents, in: B. Jacobs and J. Rutten (Eds.), *CMCS'99-Coalgebraic Methods in Computer Science*, TCS: Electronic Notes in Theoretical Computer Science, Vol. 19, Elsevier, 1999.
28. P. Wegner, D. Goldin. Interaction as a framework for modeling, in: P. Chen *et al.* (Eds.), *Conceptual Modeling - Current Issues and Future Directions*, Lecture Notes in Computer Science, Vol. 1565, Springer-Verlag, Berlin, 1999, pp 243-257.
29. P. Wegner, D.Q. Goldin. Interaction, computability, and Church's thesis, *The Computer Journal* 1999 (to appear).
30. G. Winskel. *The formal semantics of programming languages: an introduction*, The MIT Press, Cambridge (Mass.), 1993.

On the Use of Duality and Geometry in Layouts for ATM Networks

Shmuel Zaks

Department of Computer Science, Technion, Haifa, Israel.
zaks@cs.technion.ac.il, <http://www.cs.technion.ac.il/~zaks>

Abstract. We show how duality properties and geometric considerations are used in studies related to virtual path layouts of ATM networks. We concentrate on the one-to-many problem for a chain network, in which one constructs a set of paths, that enable connecting one vertex with all others in the network. We consider the parameters of load (the maximum number of paths that go through any single edge) and hop count (the maximum number of paths traversed by any single message). Optimal results are known for the cases where the routes are shortest paths and for the general case of unrestricted paths. These solutions are symmetric with respect to the two parameters of load and hop count, and thus suggest duality between these two. We discuss these dualities from various points of view. The trivial ones follow from corresponding recurrence relations and lattice paths. We then study the duality properties using trees; in the case of shortest paths layouts we use binary trees, and in the general case we use ternary trees. In this latter case we also use embedding into high dimensional spheres. The duality nature of the solutions, together with the geometric approach, prove to be extremely useful tools in understanding and analyzing layout designs. They simplify proofs of known results (like the best average case designs for the shortest paths case), enable derivation of new results (like the best average case designs for the general paths case), and improve existing results (like for the all-to-all problem).

1 Introduction

In path layouts for ATM networks pairs of nodes exchange messages along pre-defined paths in the network, termed *virtual paths*. Given a physical network, the problem is to design these paths optimally. Each such design forms a layout of paths in the network, and each connection between two nodes must consist of a concatenation of such virtual paths. The smallest number of these paths between two nodes is termed the *hop count* for these nodes, and the *load* of a layout is the maximum number of virtual paths that go through any (physical) communication line. The two principal parameters that determine the optimality of the layout are the maximum load of any communication line and the maximum hop count between any two nodes. The hop count corresponds to the time to set up a connection between the two nodes, and the load measures the size of the routing tables at the nodes.

Following the model presented in [15,5], this problem has been studied from various points of view (see also Section 8).

$\begin{array}{c} l \\ \hline h \end{array}$	1	2	3	4 ...
1	P	P	P	$P \dots$
2	P	NPC	NPC	$NPC \dots$
3	NPC	NPC	NPC	$NPC \dots$
4	NPC	NPC	NPC	$NPC \dots$
...	...	...	...	...

Fig. 1. Tractability of the one-to-all problem

The existence of a design, with given bounds on the load $\mathcal{L}$ and the hop count $\mathcal{H}$ between a given node and all the other nodes was shown to be NP-complete except for few cases. This was studied in [7], and the results - whether the problem is polynomially solvable or whether it is NP-complete - are summarized in the table depicted in Fig. 1 (a related NP-complete result was presented in [15]).

Two basic problems that have been studied are the *one-to-all* (or *broadcast*) problem (e.g., [5,13,11,6]), and the *all-to-all* problem (see, e.g., [5,11,16,17,1,6]), in which one wishes to measure the hop count from one specified node (or all nodes) in the network to all other nodes.

In this paper we focus on chain networks, with an emphasis on duality properties and the use of geometry in various analytic results. Considering a chain network, where the leftmost vertex has to be the root (the one broadcasting to all others using the virtual paths), and where each path traversed by a message must be a shortest path, a family of ordered trees $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ was presented in [13], within which an optimal solution can be found, for a chain of length $\mathcal{N}$, with $\mathcal{N}$ bounded by $\binom{\mathcal{L}+\mathcal{H}}{\mathcal{L}}$. This number, which is symmetric in $\mathcal{H}$ and $\mathcal{L}$, is also equal to the number of lattice paths from $(0,0)$ to $(\mathcal{L}, \mathcal{H})$, that use horizontal and vertical steps. Optimal bounds for this shortest path case were also derived for the average case, which also turned out to be symmetric in $\mathcal{H}$ and $\mathcal{L}$.

Considering the same problem but without the shortest path restriction, termed the *general path* case, a family of tree layouts $\mathcal{T}(\mathcal{L}, \mathcal{H})$ was introduced in [6], for a chain of length $\mathcal{N}$, not assuming that the root is located at its leftmost vertex, and with $\mathcal{N}$ bounded by $\sum_{i=0}^{\min\{\mathcal{L}, \mathcal{H}\}} 2^i \binom{\mathcal{L}}{i} \binom{\mathcal{H}}{i}$ [12]. This number, which is also symmetric in $\mathcal{H}$ and $\mathcal{L}$, is equal to the number of lattice

points within an $\mathcal{L}$ -dimensional l_1 -Sphere or radius $\mathcal{H}$, and is also equal to the number of lattice paths from $(0,0)$ to $(\mathcal{L}, \mathcal{H})$, that use horizontal, vertical or (up-)diagonal steps. The main tool in this discussion was the possibility to map layouts with load bounded by $\mathcal{L}$ and hop count bounded by $\mathcal{H}$ into this sphere.

As a consequence, the trees $\mathcal{T}(\mathcal{L}, \mathcal{H})$ and $\mathcal{T}(\mathcal{H}, \mathcal{L})$ have the same number of nodes, and so do the trees $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ and $\mathcal{T}_{short}(\mathcal{H}, \mathcal{L})$. It turns out that these dualities bear a lot of information regarding the structure of these trees, and exploring this duality, together with the use of the high dimensional spheres, proved to be extremely useful in understanding and analyzing layout designs: they simplify proofs of known results, enable derivation of new results, and improve existing results.

We use one-to-one correspondences, using binary and ternary trees, in order to combinatorially explain the duality between these two measures of hop count and load, as reflected by these above symmetries. These correspondences shed more light into the structure of these two families of trees, allowing to find for any optimal layout with $\mathcal{N}$ nodes, load $\mathcal{L}$ and minimal (or minimal average) hop count $\mathcal{H}$, its *dual* layout, having $\mathcal{N}$ nodes, maximal hop count $\mathcal{L}$ and minimal (or minimal average) load $\mathcal{H}$, and vice-versa. Moreover, they give one proof for both measures, whereas in the above-mentioned papers these symmetries were only derived as a consequence of the final result; we note that the average-case results were derived by a seemingly-different formulas, whereas the worst-case results were derived by symmetric arguments. In addition, these correspondences also provide a simple proof to a new result concerning the duality of these two parameters in the worst case and the average case analysis for the general path case layouts. Finally, it is shown that an optimal worst case solution for the shortest path and general cases, is also an optimal average case solution in both cases, allowing a simpler characterization of these optimal layouts. We then introduce the relation between high dimensional spheres and layouts for the general case. This is then used in simplifying proofs of known results, in derivation of new results (like the best average case designs for the general paths case), and in improving existing results (like for the all-to-all problem).

This survey paper is based on results presented in previous studies, as detailed in the following description of its structure. In Section 2 the ATM model is presented, following [5]. In Section 3 we discuss the optimal solutions; the optimal design for the shortest path case follows the discussion in [13], and the optimal design for the general case follows the discussion in [6,8]. We encounter the duality of the parameters of load and hop count, which follows via recurrence relations. In Section 4 we discuss relations with lattice paths. In Section 5 we describe the use of binary and ternary trees to shed more direct light on these duality results, and the use of high dimensional spheres is discussed in Section 6, both following ([6,8]). The applications of the tools of duality and geometry are presented in Section 7, following [8]). We close with a discussion in Section 8.

2 The Model

We model the underlying communication network as an undirected graph $G = (V, E)$, where the set V of vertices corresponds to the set of switches, and the set E of edges corresponds to the physical links between them.

Definition 1. A rooted virtual path layout (*layout* for short) Ψ is a collection of simple paths in G , termed virtual paths (*VPs* for short), and a vertex $r \in V$ termed the root of the layout (denoted $\text{root}(\Psi)$).

Definition 2. The load $\mathcal{L}(e)$ of an edge $e \in E$ in a layout Ψ is the number of VPs $\psi \in \Psi$ that include e .

Definition 3. The load $\mathcal{L}_{\max}(\Psi)$ of a layout Ψ is $\max_{e \in E} \mathcal{L}(e)$.

Definition 4. The hop count $\mathcal{H}(v)$ of a vertex $v \in V$ in a layout Ψ is the minimum number of VPs whose concatenation forms a path in G from v to $\text{root}(\Psi)$. If no such VPs exist, define $\mathcal{H}(v) = \infty$.

Definition 5. The maximal hop count of Ψ is $\mathcal{H}_{\max}(\Psi) = \max_{v \in V} \{\mathcal{H}(v)\}$.

In the rest of this paper we assume that the underlying network is a *chain*. We consider two cases: the one in which only shortest paths are allowed, and the second one in which general paths are considered.

To minimize the load, one can use a layout Ψ which has a VP on each physical link, i.e., $\mathcal{L}_{\max}(\Psi) = 1$, however such a layout has a hop count of $\mathcal{N} - 1$. The other extreme is connecting a direct VP from the root to each other vertex, yielding $\mathcal{H}_{\max} = 1$, but then $\mathcal{L}_{\max} = \mathcal{N} - 1$. For the intermediate cases we need the following definitions.

Definition 6. $\mathcal{H}_{\text{opt}}(\mathcal{N}, \mathcal{L})$ denotes the optimal hop count of any layout Ψ on a chain of $\mathcal{N}$ vertices such that $\mathcal{L}_{\max}(\Psi) \leq \mathcal{L}$, i.e.,

$$\mathcal{H}_{\text{opt}}(\mathcal{N}, \mathcal{L}) \equiv \min_{\Psi} \{ \mathcal{H}_{\max}(\Psi) : \mathcal{L}_{\max}(\Psi) \leq \mathcal{L} \}.$$

Definition 7. $\mathcal{L}_{\text{opt}}(\mathcal{N}, \mathcal{H})$ denotes the optimal load of any layout Ψ on a chain of $\mathcal{N}$ vertices such that $\mathcal{H}_{\max}(\Psi) \leq \mathcal{H}$, i.e.,

$$\mathcal{L}_{\text{opt}}(\mathcal{N}, \mathcal{H}) \equiv \min_{\Psi} \{ \mathcal{L}_{\max}(\Psi) : \mathcal{H}_{\max}(\Psi) \leq \mathcal{H} \}.$$

Definition 8. Two VPs constitute a crossing if their endpoints l_1, l_2 and r_1, r_2 satisfy $l_1 < l_2 < r_1 < r_2$. A layout is called crossing-free if no pair of VPs constitute a crossing.

It is known ([13,1]) that for each performance measure ($\mathcal{L}_{max}$, $\mathcal{H}_{max}$, $\mathcal{L}_{avg}$, $\mathcal{H}_{avg}$) there exists an optimal layout which is crossing-free. In the rest of the paper we restrict ourselves to layouts viewed as a planar (that is, crossing-free) embedding of a tree on the chain, also termed *tree layouts*. Therefore, when no confusion occurs, we refer to each VP in a given layout Ψ an *edge* of Ψ .

$\mathcal{N}_{short}(\mathcal{L}, \mathcal{H})$ denotes the length of a longest chain in which one node can broadcast to all others, with at most $\mathcal{H}$ hops and a load bounded by $\mathcal{L}$, for the case of shortest paths. The similar measure for the general case is denoted by $\mathcal{N}(\mathcal{L}, \mathcal{H})$.

3 Optimal Solutions and Their Duality

In this section we present the optimal solutions for layouts, when messages have to travel either along shortest paths or general paths. We'll show the symmetric role played by the load and hop count, and explain it via the corresponding recurrence relations.

3.1 Optimal Virtual Path for the Shortest Path Case

Assuming that the leftmost node in the chain has to broadcast to each node to its right, it is clear that, for given $\mathcal{H}$ and $\mathcal{L}$, the largest possible chain for which such a design exists is like the one shows in Fig. 2.

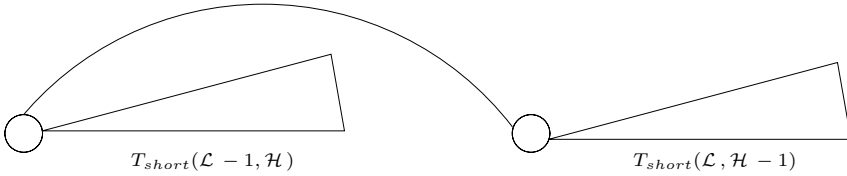


Fig. 2. The tree layout $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$

The design depicted in Fig. 2 uses the trees $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ defined as follows.

Definition 9. The tree layout $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ is defined recursively as follows. $\mathcal{T}_{short}(\mathcal{L}, 0)$ and $\mathcal{T}_{short}(0, \mathcal{H})$ are tree layouts with a unique node. Otherwise, the root of a tree layout $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ is the leftmost node of a $\mathcal{T}_{short}(\mathcal{L} - 1, \mathcal{H})$ tree layout, and it is also the leftmost node of a tree layout $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H} - 1)$

Recall that $\mathcal{M}_{short}(\mathcal{L}, \mathcal{H})$ is the length of the longest chain in which a design exists, for a broadcast from the leftmost node to all others, for given parameters $\mathcal{H}$ and $\mathcal{L}$. $\mathcal{M}_{short}(\mathcal{L}, \mathcal{H})$ clearly satisfies the following recurrence relation:

$$\begin{aligned} \mathcal{M}_{short}(0, \mathcal{H}) &= \mathcal{M}_{short}(\mathcal{L}, 0) = 1 \quad \forall \mathcal{H}, \mathcal{L} \geq 0 \\ \mathcal{M}_{short}(\mathcal{L}, \mathcal{H}) &= \mathcal{M}_{short}(\mathcal{L}, \mathcal{H} - 1) + \mathcal{M}_{short}(\mathcal{L} - 1, \mathcal{H}) \quad \forall \mathcal{H}, \mathcal{L} > 0 . \end{aligned} \tag{1}$$

It easily follows that

$$\mathcal{M}_{\text{short}}(\mathcal{L}, \mathcal{H}) = \binom{\mathcal{L} + \mathcal{H}}{\mathcal{H}}. \quad (2)$$

The expression in 2 is clearly symmetric in $\mathcal{H}$ and $\mathcal{L}$, which establishes the first result in which the load and hop count play symmetric roles.

Note that it is clear that the maximal number $\mathcal{N}_{\text{short}}(\mathcal{L}, \mathcal{H})$ of nodes in a chain to which one node can broadcast using shortest paths, satisfies

$$\mathcal{N}_{\text{short}}(\mathcal{L}, \mathcal{H}) = 2 \binom{\mathcal{L} + \mathcal{H}}{\mathcal{H}} - 1.$$

Using these trees, it is easy to show that $\mathcal{L}_{\text{max}}(\mathcal{T}_{\text{short}}(\mathcal{L}, \mathcal{H})) = \mathcal{L}$ and $\mathcal{H}_{\text{max}}(\mathcal{T}_{\text{short}}(\mathcal{L}, \mathcal{H})) = \mathcal{H}$. The following two theorems follow:

Theorem 1. *Consider a chain of $\mathcal{N}$ vertices and a maximal load requirement $\mathcal{L}$. Let $\mathcal{H}$ be such that*

$$\binom{\mathcal{L} + \mathcal{H} - 1}{\mathcal{L}} < \mathcal{N} \leq \binom{\mathcal{L} + \mathcal{H}}{\mathcal{L}}.$$

Then $\mathcal{H}_{\text{opt}}(\mathcal{N}, \mathcal{L}) = \mathcal{H}$.

Theorem 2. *Consider a chain of $\mathcal{N}$ vertices and a maximal hop requirement $\mathcal{H}$. Let $\mathcal{L}$ be such that*

$$\binom{\mathcal{L} + \mathcal{H} - 1}{\mathcal{H}} < \mathcal{N} \leq \binom{\mathcal{L} + \mathcal{H}}{\mathcal{H}}.$$

Then $\mathcal{L}_{\text{opt}}(\mathcal{N}, \mathcal{H}) = \mathcal{L}$.

Optimal bounds were also derived in [13,14] for the average case, using dynamic programming; the results use different recursive constructions, but end up in structures that are symmetric in $\mathcal{H}$ and $\mathcal{L}$. These results are stated as follows:

Theorem 3. *Let n and $\mathcal{H}$ be given. Let $\mathcal{L}$ be the largest integer such that $N \geq \binom{\mathcal{L} + \mathcal{H}}{\mathcal{L}}$, and let $r = N - \binom{\mathcal{L} + \mathcal{H}}{\mathcal{L}}$. Then*

$$\mathcal{L}_{\text{tot}}(N, \mathcal{H}) = \mathcal{H} \binom{\mathcal{L} + \mathcal{H}}{\mathcal{L} - 1} + r(\mathcal{L} + 1).$$

Theorem 4. *Let N and $\mathcal{L}$ be given. Let $\mathcal{H}$ be the maximal such that $N \geq \binom{\mathcal{L} + \mathcal{H}}{\mathcal{H}}$, and let $r = N - \binom{\mathcal{L} + \mathcal{H}}{\mathcal{H}}$. Then*

$$\mathcal{H}_{\text{tot}}(N, \mathcal{L}) = \mathcal{L} \binom{\mathcal{L} + \mathcal{H}}{\mathcal{H} - 1} + r(\mathcal{H} + 1).$$

Note that these last two theorems imply that in $\mathcal{T}_{\text{short}}(\mathcal{L}, \mathcal{H})$ the average hop is $\frac{\mathcal{L}}{\mathcal{L}+1}\mathcal{H}$, and the average load is slightly smaller than $\frac{\mathcal{H}}{\mathcal{H}+1}\mathcal{L}$.

3.2 Optimal Virtual Path for the General Case

In the case where not only shortest paths are traversed, a new family of optimal tree layouts $\mathcal{T}(\mathcal{L}, \mathcal{H})$ is now presented.

Definition 10. The tree layout $\mathcal{T}(\mathcal{L}, \mathcal{H})$ is defined recursively as follows.

$\mathcal{T}_{right}(\mathcal{L}, 0)$, $\mathcal{T}_{right}(0, \mathcal{H})$, $\mathcal{T}_{left}(\mathcal{L}, 0)$ and $\mathcal{T}_{left}(0, \mathcal{H})$ are tree layouts with a unique node. Otherwise, the root r is also the rightmost node of a tree layout $\mathcal{T}_{right}(\mathcal{L}, \mathcal{H})$ and the leftmost node of a tree layout $\mathcal{T}_{left}(\mathcal{L}, \mathcal{H})$, when the tree layouts $\mathcal{T}_{left}(\mathcal{L}, \mathcal{H})$ and $\mathcal{T}_{right}(\mathcal{L}, \mathcal{H})$ are also defined recursively as follows. The root of a tree layout $\mathcal{T}_{left}(\mathcal{L}, \mathcal{H})$ is the leftmost node of a $\mathcal{T}_{left}(\mathcal{L} - 1, \mathcal{H})$ tree layout, and it is also connected to a node which is the root of a tree layout $\mathcal{T}_{right}(\mathcal{L} - 1, \mathcal{H} - 1)$ and a tree layout $\mathcal{T}_{left}(\mathcal{L}, \mathcal{H} - 1)$ (see Fig. 3). Note that the root of $\mathcal{T}_{left}(\mathcal{L}, \mathcal{H})$ is its leftmost node. The tree layout $\mathcal{T}_{right}(\mathcal{L}, \mathcal{H})$ is defined as the mirror image of $\mathcal{T}_{left}(\mathcal{L}, \mathcal{H})$.

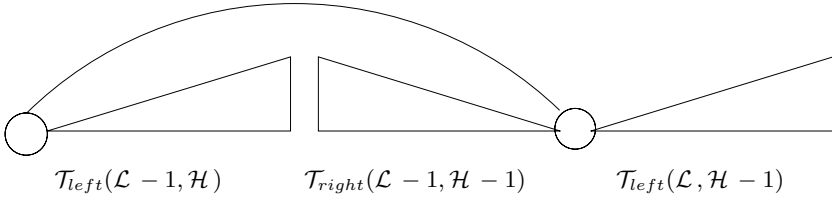


Fig. 3. $\mathcal{T}_{left}(\mathcal{L}, \mathcal{H})$ recursive definition

Denote by $\mathcal{N}(\mathcal{L}, \mathcal{H})$ the longest chain in which it is possible to connect one node to all others, with at most $\mathcal{H}$ hops and the load bounded by $\mathcal{L}$. From the above, it is clear that this chain is constructed from two chains as above, glued at their root. $\mathcal{N}(\mathcal{L}, \mathcal{H})$ clearly satisfies the following recurrence relation:

$$\mathcal{N}(0, \mathcal{H}) = \mathcal{N}(\mathcal{L}, 0) = 1 \quad \forall \mathcal{H}, \mathcal{L} \geq 0 \quad (3)$$

$$\mathcal{N}(\mathcal{L}, \mathcal{H}) = \mathcal{N}(\mathcal{L}, \mathcal{H} - 1) + \mathcal{N}(\mathcal{L} - 1, \mathcal{H}) + \mathcal{N}(\mathcal{L} - 1, \mathcal{H} - 1) \quad \forall \mathcal{H}, \mathcal{L} > 0.$$

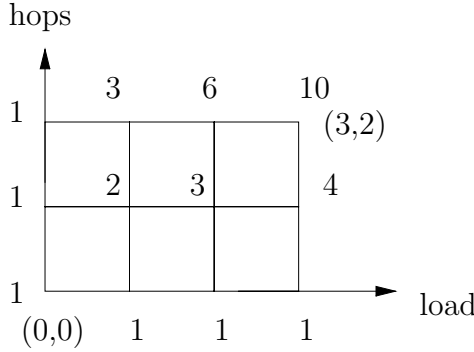
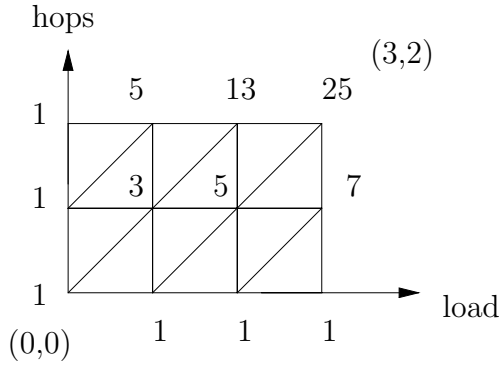
Again, the symmetric role of the hop count and the load are clear both from the definition of the corresponding trees and from the recurrence relations that compute their sizes.

It is known ([12]) that the solution to the recurrence relation (3) is given by

$$\mathcal{N} = \sum_{i=0}^{\min\{\mathcal{L}, \mathcal{H}\}} 2^i \binom{\mathcal{L}}{i} \binom{\mathcal{H}}{i}. \quad (4)$$

4 Correspondences with Lattice Paths

The recurrence relation (1) clearly corresponds to the number of lattice paths from the point (0,0) to the point $(\mathcal{L}, \mathcal{H})$, that use only horizontal (right) and vertical (up) steps.

**Fig. 4.** Lattice paths with regular steps**Fig. 5.** Lattice paths with regular and diagonal steps

In Fig. 4 each lattice point is labeled with the number of lattice paths from $(0,0)$ to it; the calculation is done by the recurrence relation 1. For the case $\mathcal{L} = 3$ and $\mathcal{H} = 2$ one gets $\binom{3+2}{2} = 10$; this corresponds to the number of nodes in the tree $\mathcal{T}_{short}(3,2)$ (see Fig. 6), and to the number of paths that go from $(0,0)$ to $(3,2)$.

The recurrence relation (3) clearly corresponds to the number of lattice paths from the point $(0,0)$ to the point $(\mathcal{L}, \mathcal{H})$, that use horizontal (right), vertical (up), and diagonal (up-right) steps. In Fig. 5 each lattice point is labeled with the number of lattice paths from $(0,0)$ to it. For the case $\mathcal{L} = 3$ and $\mathcal{H} = 2$ one gets 25 such paths. This corresponds to the number of nodes in the tree $\mathcal{T}(3,2)$

that is constructed of two trees, glued at their roots, the one $(\mathcal{T}_{left}(3,2))$ depicted in Fig. 6 (and containing 13 vertices), and its corresponding reverse tree.

We also refer to these lattice paths in Section 6.

5 Duality: Binary Trees and Ternary Trees

We saw in Section 3 that the layouts $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ and $\mathcal{T}_{short}(\mathcal{H}, \mathcal{L})$ and also $\mathcal{T}(\mathcal{L}, \mathcal{H})$ and $\mathcal{T}(\mathcal{H}, \mathcal{L})$ have the same number of vertices. We now turn to show that each pair within these virtual path layouts are, actually, quite strongly related. In Section 5.1 we deal with layouts that use shortest-length paths, and show their close relations to a certain class of binary trees, and in Section 5.2 we deal with the general layouts and show their close relations to a certain class of ternary trees.

5.1 $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ and Binary Trees

In this section we show how to transform any layout Ψ with hop count bounded by $\mathcal{H}$ and load bounded by $\mathcal{L}$ for layouts using only shortest paths, into a layout $\bar{\Psi}$ (its *dual*) with hop count bounded by $\mathcal{L}$ and load bounded by $\mathcal{H}$. In particular, this mapping will transform $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ into $\mathcal{T}_{short}(\mathcal{H}, \mathcal{L})$.

To show this, we use transformation between any layout with x virtual paths (depicted as edges) and binary trees with x nodes (in a binary tree, each internal node has a left child and/or a right child). We'll derive our main correspondence between $\mathcal{T}_{short}(\mathcal{H}, \mathcal{L})$ and $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ for $x = \mathcal{N} - 1$, where $\mathcal{N} = \binom{\mathcal{L} + \mathcal{H}}{\mathcal{L}}$. Our correspondence is done in three steps, as follows.

Step 1: Given a planar layout Ψ we transform it into a binary tree $T = b(\Psi)$, under which each edge e is mapped to a node $b(e)$, as follows. Let $e = (r, v)$ be the edge outgoing the root r to the rightmost vertex (to which there is a VP; we call this a *1-level* edge). This edge e is mapped to the root $b(r)$ of T . Remove e from Ψ . As a consequence, two layouts remain: Ψ_1 with root r and Ψ_2 with root v , when their roots are located at the leftmost vertices of both layouts. Recursively, the left child of node $b(e)$ will be $b(\Psi_1)$ and its right child will be $b(\Psi_2)$. If any of the layouts Ψ is empty, so is its image $b(\Psi)$ (in other words, we can stop when a Ψ that consists of a single edge is mapped to a binary tree that consists of a single vertex).

Step 2: Build a binary tree $\bar{T}$, which is a reflection of T (that is, we exchange the left child and the right child of each vertex).

Step 3: We transform back the binary tree $\bar{T}$ into the (unique) layout $\bar{\Psi}$ such that $b(\bar{\Psi}) = \bar{T}$

In Fig. 6 the layouts for $\mathcal{L} = 2, \mathcal{H} = 3$ and $\mathcal{L} = 3, \mathcal{H} = 2$ are shown, together with the corresponding trees $\mathcal{T}_{short}(2, 3)$ and $\mathcal{T}_{short}(3, 2)$, and the corresponding binary trees constructed as explained above. The edge e in the layout $\mathcal{T}_{short}(3, 2)$ is assigned the vertex $b(e)$ in the corresponding tree $b(\mathcal{T}_{short}(3, 2))$.

Given a non-crossing layout Ψ , we define the *level* of an edge e in Ψ , denoted $level_{\Psi}(e)$ (or $level(e)$ for short), to be one plus the number of edges above e in Ψ . In addition, to each edge e of the layout Ψ we assign its farthest end-point from the root, $v(e)$.

In Fig. 6 the edge e in the layout $\mathcal{T}_{short}(3, 2)$ is assigned the vertex $v(e)$ in this layout, and its level $level(e)$ is 2.

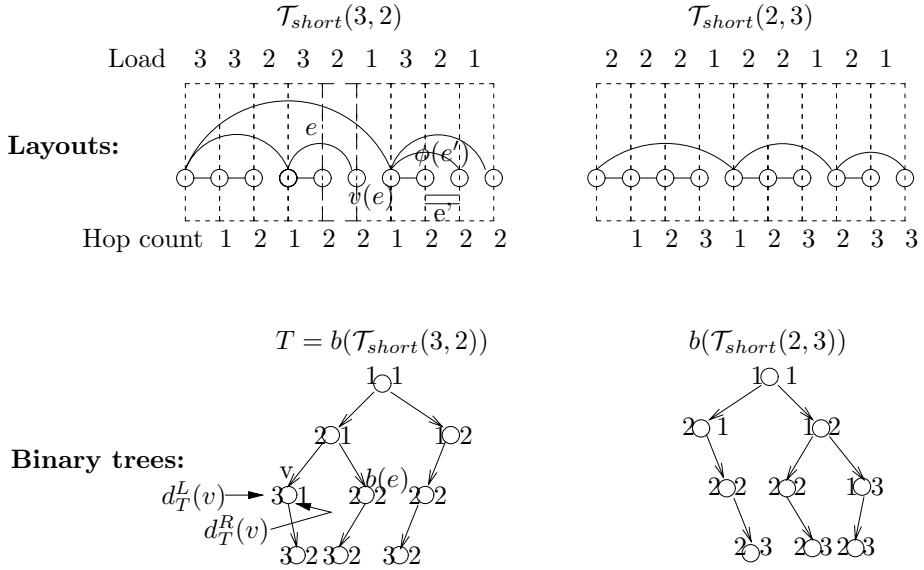


Fig. 6. An example of the transformation using binary trees

One of our key observations is the following theorem:

Theorem 5. *For every $\mathcal{H}$ and $\mathcal{L}$, the trees $b(\mathcal{T}_{short}(\mathcal{L}, \mathcal{H}))$ and $b(\mathcal{T}_{short}(\mathcal{H}, \mathcal{L}))$ are reflections of each other.*

This clearly establishes a one-to-one mapping between these trees, and thus establishes the required duality.

To further investigate the structure of these trees, we now turn to explore the properties of the binary trees that we have defined above. We prove the following theorem:

Theorem 6. *Given a layout Ψ , let $T = b(\Psi)$ be the binary tree assigned to it by the transformation above. Let $d_T^L(v)$ ($d_T^R(v)$) be equal to one plus the number of left (right) steps in the path from the root r to v , for every node v in T . Then, for every edge e in the layout Ψ :*

1. $\mathcal{H}_\Psi(v(e)) = d_T^R(b(e))$, and
2. $level(e) = d_T^L(b(e))$.

Given a non-crossing layout Ψ , for each physical link e' we assign an edge $\phi(e')$ in Ψ that includes it and is of highest level (such a path exists due to the connectivity and planarity of the layout; see edge e' and physical edge $\phi(e')$ in Fig. 6). It can be easily proved that:

Lemma 1. *Given a non-crossing tree layout Ψ , the mapping of a physical link e' to an edge $\phi(e')$ described above is one-to-one.*

Proposition 1. *Given a non-crossing tree layout Ψ over a physical network, let $T = b(\Psi)$ be the binary tree assigned to it. Then $\mathcal{L}(e') = \text{level}(\phi(e'))$ for every edge e' in the physical network.*

Given a layout Ψ over a chain network, if we consider the multiset $\{d_T^R(v) | v \in b(\Psi)\}$ we get exactly the multiset of hop counts of the vertices of this network (by Theorem 6), and if we consider the multiset $\{d_T^L(v) | v \in b(\Psi)\}$ we get exactly the multiset of loads of the physical links of this network (by Theorem 6 and Proposition 1). By using this and finding the dual layout $\bar{\Psi}$ with the multisets $\{d_T^R(v) | v \in b(\bar{\Psi})\}$ of hop counts of its vertices and $\{d_T^L(v) | v \in b(\bar{\Psi})\}$ of loads of its physical edges of $\bar{\Psi}$, we observe that the multiset of hop counts of $\bar{\Psi}$ is exactly the multiset of load of Ψ , and the multiset of loads of $\bar{\Psi}$ is also the multiset of hop counts of Ψ , thus deriving a complete combinatorial explanation for the symmetric results of Section 3.1 for either the worst case trees or average case trees:

Theorem 7. *Given an optimal layout Ψ with $\mathcal{N}$ nodes, load bounded by $\mathcal{L}$ and optimal hop count $\mathcal{H}_{\text{opt}}(\mathcal{N}, \mathcal{L})$, its dual layout $\bar{\Psi}$ has $\mathcal{N}$ nodes, hop count bounded by $\mathcal{L}$ and optimal load $\mathcal{H}_{\text{opt}}(\mathcal{N}, \mathcal{L})$.*

Theorem 8. *Given an optimal layout Ψ with $\mathcal{N}$ nodes, hop count bounded by $\mathcal{H}$ and optimal load $\mathcal{L}_{\text{opt}}(\mathcal{N}, \mathcal{H})$, its dual layout $\bar{\Psi}$ has $\mathcal{N}$ nodes, load bounded by $\mathcal{H}$ and optimal hop count $\mathcal{L}_{\text{opt}}(\mathcal{N}, \mathcal{H})$.*

Theorem 9. *Given an optimal layout Ψ with $\mathcal{N}$ nodes, load bounded by $\mathcal{L}$ and optimal average hop count, its dual layout $\bar{\Psi}$ has $\mathcal{N}$ nodes, hop count bounded by $\mathcal{L}$ and optimal average load.*

Theorem 10. *Given an optimal layout Ψ with $\mathcal{N}$ nodes, hop count bounded by $\mathcal{H}$ and optimal average load, its dual layout $\bar{\Psi}$ has $\mathcal{N}$ nodes, load bounded by $\mathcal{H}$ and optimal average hop count.*

5.2 $\mathcal{T}(\mathcal{L}, \mathcal{H})$ and Ternary Trees

We now extend the technique developed in Section 5.1 to general path case layouts; we show how to transform any layout Ψ with hop count bounded by $\mathcal{H}$ and load bounded by $\mathcal{L}$ into a layout $\bar{\Psi}$ (its *dual*) with hop count bounded by $\mathcal{L}$ and load bounded by $\mathcal{H}$. In particular, this mapping will transform $\mathcal{T}(\mathcal{L}, \mathcal{H})$ into $\mathcal{T}(\mathcal{H}, \mathcal{L})$.

To show this, we use transformation between any layout with x edges (VPs) and ternary trees with x nodes (in a ternary tree, each internal node has a left child and/or a middle child and/or a right child). Our correspondence is done in three steps, as follows.

Step 1: Given a planar layout Ψ we transform it into a ternary tree $T = t(\Psi)$, under which each edge e is mapped to a node $t(e)$, as follows. Let $e = (r, v)$ be

the edge outgoing the root r to the rightmost vertex (to which there is a VP; we call this a *1-level* edge). This edge e is mapped to the root $t(r)$ of T . Remove e from Ψ . As a consequence, three layouts remain: Ψ_1 with root r and Ψ_3 with root v (when their roots are located at the leftmost vertices of both layouts) and Ψ_2 with root v (when v is its rightmost vertex). Recursively, the left child of node $t(e)$ will be $t(\Psi_1)$, its middle child will be $t(\Psi_2)$ and its right child will be $t(\Psi_3)$. If any of the layouts Ψ is empty, so is its image $t(\Psi)$ (in other words, we can stop when a Ψ that consists of a single edge is mapped to a ternary tree that consists of a single vertex).

Step 2: Build a ternary tree $\bar{T}$, which is a reflection of T (that is, we exchange the left child and the right child of each vertex; the middle child does not change).

Step 3: We transform back the ternary tree $\bar{T}$ into the (unique) layout $\bar{\Psi}$ such that $t(\bar{\Psi}) = \bar{T}$.

See Fig. 7 for an example of this transformation.

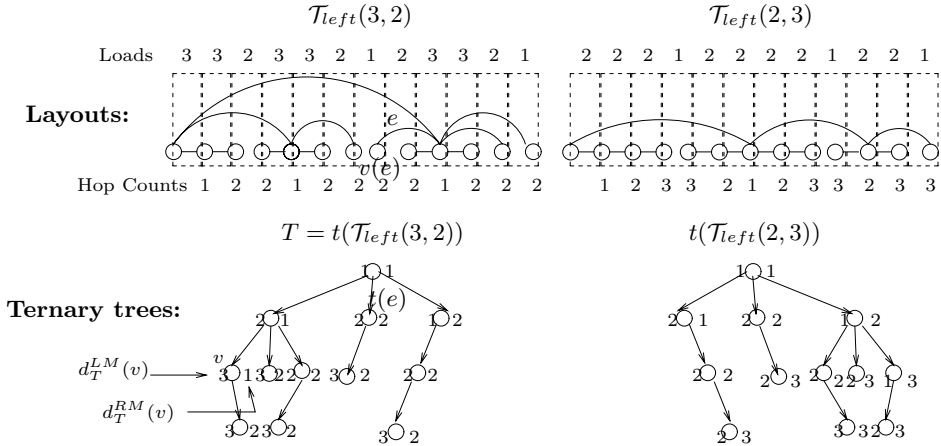


Fig. 7. An example of the transformation using ternary trees

One of our key observations is the following theorem:

Theorem 11. *For every $\mathcal{H}$ and $\mathcal{L}$, the trees $t(\mathcal{T}(\mathcal{L}, \mathcal{H}))$ and $t(\mathcal{T}(\mathcal{H}, \mathcal{L}))$ are reflections of each other.*

This clearly establishes a one-to-one mapping between these trees, and thus establishes the required duality.

To further investigate the structure of these trees, we now turn to explore the properties of the ternary trees that we have defined above. We prove the following theorem. Note that the definition of *level* (of an edge) and ϕ (of a physical link) remain exactly the same as in Section 5.1.

Theorem 12. *Given a layout Ψ , let $T = t(\Psi)$ be the ternary tree assigned to it by the transformation above. Let $d_T^{LM}(v)$ ($d_T^{RM}(v)$) be equal to one plus the*

number of left and middle (right and middle) steps in the path from the root r to v , for every node v in T . Then, for every edge e in the layout Ψ :

1. $\mathcal{H}\Psi(v(e)) = d_T^{RM}(t(e))$, and
2. $\text{level}(e) = d_T^{LM}(t(e))$.

Proposition 2. *Given a non-crossing tree layout Ψ over a physical network, let $T = t(\Psi)$ be the ternary tree assigned to it. Then $\mathcal{L}(e') = \text{level}(\phi(e'))$ for every edge e' in the physical network.*

Given a layout Ψ over a chain network, if we consider the multiset $\{d_T^{RM}(v) | v \in t(\Psi)\}$ we get exactly the multiset of hop counts of the vertices of this network (by Theorem 12), and if we consider the multiset $\{d_T^{LM}(v) | v \in t(\Psi)\}$ we get exactly the multiset of loads of the physical links of this network (by Theorem 12 and Proposition 2). By using this and finding the dual layout $\bar{\Psi}$ with the multisets $\{d_T^{RM}(v) | v \in t(\bar{\Psi})\}$ of hop counts of its vertices and $\{d_T^{LM}(v) | v \in t(\bar{\Psi})\}$ of loads of its physical edges of $\bar{\Psi}$, we observe that the multiset of hop counts of $\bar{\Psi}$ is exactly the multiset of load of Ψ , and the multiset of loads of $\bar{\Psi}$ is also the multiset of hop counts of Ψ , thus deriving a complete combinatorial explanation for the symmetric results of either the worst-case trees or average-case trees in the general path case.

Following the above discussion, we obtain the exact four theorems (Theorems 7, 8, 9 and 10) extended to the general path case layouts.

6 Use of Geometry

Consider the set $Sp(\mathcal{L}, \mathcal{H})$ of lattice points (that is, points with integral coordinates) of an $\mathcal{L}$ -dimensional l_1 -Sphere of radius $\mathcal{H}$. The points in this sphere are $\mathcal{L}$ -dimensional vectors $\mathbf{v} = (v_1, v_2, \dots, v_{\mathcal{L}})$, where $|v_1| + |v_2| + \dots + |v_{\mathcal{L}}| \leq \mathcal{H}$. Let $Rad(N, \mathcal{L})$ be the radius of the smallest $\mathcal{L}$ -dimensional l_1 -Sphere containing at least N internal lattice points. For example, $Sp(1, 2)$ contains 5 lattice points, and $Rad(6, 2) = 3$.

We show that

Theorem 13. *The tree $\mathcal{T}(\mathcal{L}, \mathcal{H})$ contains $|Sp(\mathcal{L}, \mathcal{H})|$ vertices.*

The exact number of points in this sphere is given by equation (4). (This was studied, in connection with codewords, in [12].)

Moreover, we can show that

Theorem 14. *Consider a chain of N vertices and a maximal load requirement $\mathcal{L}$. Then $\mathcal{H}_{opt}(N, \mathcal{L}) = Rad(N, \mathcal{L})$.*

These theorems are proved by showing a one-to-one mapping between the nodes of any layout with hop count bounded by $\mathcal{H}$ and load bounded by $\mathcal{L}$ into $Sp(\mathcal{L}, \mathcal{H})$. This mapping turns out to be a very useful tool in derivations of analytical results (see Section 8). The details of this embedding can be found

in [6,8]. A short description of this embedding is now described, followed by an example.

In this embedding, a node for which the hop count is h will be mapped onto a point $(x_1, x_2, \dots, x_{\mathcal{L}})$, such that $|x_1| + |x_2| + \dots + |x_{\mathcal{L}}| = h$. This embedding starts by mapping the root of the given layout onto the origin $(0, 0, \dots, 0)$. The algorithm continues in $\mathcal{L}$ phases. In the first phase we consider the paths, emanating from the root in both directions. The nodes on one path are mapped to the points $(1, 0, \dots, 0)$, $(2, 0, \dots, 0)$, and so on, and the nodes on the other path to the points $(-1, 0, \dots, 0)$, $(-2, 0, \dots, 0)$, and so on. In each subsequent phase, we continue in the same manner from each node that already got mapped, and for each such node, the new nodes are mapped to points that differ from it in the second component.

We present now an example of this embedding. We illustrate our algorithm on the tree layout T shown in Figure 8(A). $first(T) = a$, $last(T) = d$, and $root = c$. The path P_1 is $first(T) = a - b - c = root$ and the path P_2 is $root = c - d = last(T)$. We thus map in the first stage ($\xi = 1$) the nodes a, b, c and d to the points $(2, 0, 0)$, $(1, 0, 0)$, $(0, 0, 0)$ and $(-1, 0, 0)$, respectively (see Figure 8(B)). We then delete these edges from T , and the remaining graph (forest) is shown in Figure 8(C). At this 2nd stage ($\xi = 2$), the nodes b, c and d are roots of non-trivial layouts, and the algorithm maps the nodes e, f , and g to the points $(1, -1, 0)$, $(0, -1, 0)$ and $(-1, 1, 0)$, respectively. Note that $LABEL(e)[1] = LABEL(b)[1] = 1$. The corresponding edges are then deleted from the layout, and we result in the graph depicted in Figure 8(D), which results in a similar mapping for nodes h and i .

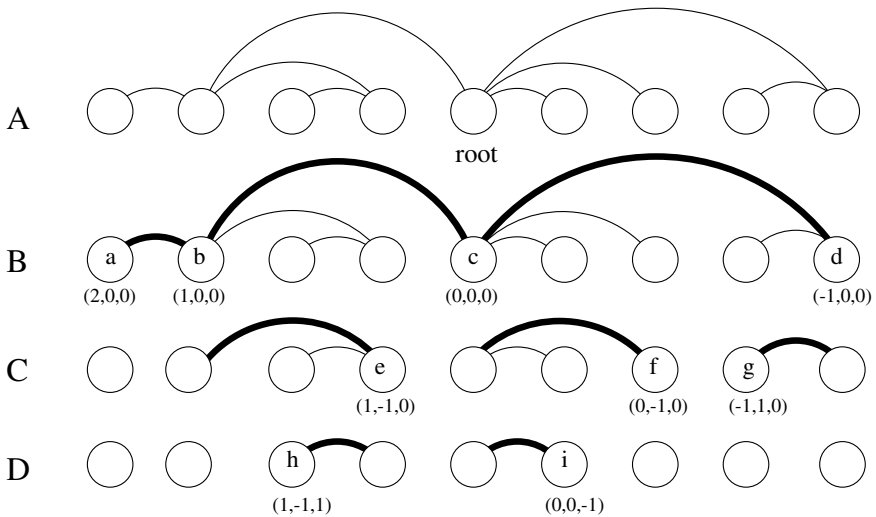


Fig. 8. Embedding of a tree layout of load 3

We now sketch a one-to-one mapping between the set of lattice points of the $\mathcal{L}$ -dimensional sphere of radius $\mathcal{H}$ and the set of lattice paths from $(0, 0)$ to $(\mathcal{L}, \mathcal{H})$ that use horizontal, vertical or (up-)diagonal steps. We first describe a function which maps every vector $\mathbf{v} = (v_1, \dots, v_{\mathcal{L}})$ in $Sp(\mathcal{L}, \mathcal{H})$ into such a lattice path. Starting from $(0, 0)$ make $|v_1|$ vertical steps and one horizontal step, make $|v_2|$ vertical steps and one horizontal step, ..., make $|v_{\mathcal{L}}|$ vertical steps and one more horizontal step, ending with $\mathcal{H} - \sum_{i=1}^{\mathcal{L}} v_i$ horizontal steps. After that, for every negative v_i component of $\mathbf{v}$, we replace the $|v_i|$ th vertical step and the subsequent horizontal step done during the translation of this component by an (up-)diagonal step. A close look at the properties of these paths enables us to further explore the properties of these trees. Returning to the discussion of the layouts $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ that use only shortest paths, it is possible to find a similar correspondence between the vertices of these trees and lattice paths from $(0, 0)$ to $(\mathcal{L}, \mathcal{H})$ that use only vertical and horizontal steps, and to view some properties of these trees using these lattice paths.

7 Applications

The insight gained by the duality properties of the solutions for both the shortest path case and the general path case, and the one-to-one correspondence between layouts with a hop count bounded by $\mathcal{H}$ and load bounded by $\mathcal{L}$ with the lattice points in $Sp(\mathcal{L}, \mathcal{H})$, have proved to be quite powerful in deriving analytical results.

1. Using the insight we got for the trees $\mathcal{T}_{short}(\mathcal{L}, \mathcal{H})$ due to the duality between the hop count and the load, we managed to supply very short proofs for the optimal average hop count and load in the shortest path case, as detailed in Theorems 3 and 4 (for details, see [8]). Moreover, the duality properties imply that a solution for a certain setting of the parameters implies a solution for the same setting, where the roles of the hop count and the load are interchanged. See Theorems 7, 8, 9 and 10, and the last sentence of Section 5.
2. Using the duality properties, and especially using the high dimensional spheres, the following theorem can be proved (see [8]), regarding the optimal average hop count and load in the general path case.

Theorem 15. *Let N and $\mathcal{H}$ be given. Let $\mathcal{L}$ be the maximal l such that $|Sp(l, \mathcal{H})| \leq N$, and let $r = N - |T(\mathcal{L}, \mathcal{H})|$. Then*

$$\mathcal{L}_{tot}(N, \mathcal{H}) = (\mathcal{L} + \frac{1}{2})|Sp(\mathcal{L}, \mathcal{H})| - \frac{1}{2}|Sp(\mathcal{L}, \mathcal{H} + 1)| + r(\mathcal{L} + 1).$$

Theorem 16. *Let N and $\mathcal{L}$ be given. Let $\mathcal{H}$ be the maximal h such that $|Sp(\mathcal{L}, h)| \leq N$, and let $r = N - |T(\mathcal{L}, \mathcal{H})|$. Then*

$$\mathcal{H}_{tot}(N, \mathcal{H}) = (\mathcal{H} + \frac{1}{2})|Sp(\mathcal{L}, \mathcal{H})| - \frac{1}{2}|Sp(\mathcal{L} + 1, \mathcal{H})| + r(\mathcal{H} + 1).$$

3. Using the above correspondences and discussion, it can be shown that the layout that we managed to design - for a given N and $\mathcal{L}$ - with an optimal worst case hop count is also optimal with respect to the average case hop count, and the layout that we managed to design - for a given N and $\mathcal{H}$ - with an optimal worst case load is also optimal with respect to the average case load. This holds for either the shortest path designs or the general path designs (for details, see [8]).
4. Using volumes of high-dimensional polyhedra, we show a trade-off between the hop count and load, as follows (for a detailed discussion, see [6,8]):

Theorem 17. *For all $\mathcal{L}$ and N :*

$$\begin{aligned} \max\{1/2 \cdot (\mathcal{L}!N)^{\frac{1}{\mathcal{L}}} - \mathcal{L}/2, 1/2 \cdot N^{\frac{1}{\mathcal{L}}} - 1/2, \frac{\log N}{\log(2 \cdot \mathcal{L} + 1)}\} \leq \\ \leq \text{Rad}(N, \mathcal{L}) < 1/2 \cdot (\mathcal{L}!N)^{\frac{1}{\mathcal{L}}} + 1/2 \end{aligned}$$

5. While the one-to-one problem is naturally related to the radius of a network, the all-to-all problem is related to its diameter. By using the fact that the diameter lies between the radius and twice the radius, and using the approximation to $\text{Rad}(N, \mathcal{L})$ as discussed above, we manage to significantly improve results regarding the all-to-all problem, presented in [16,17,1]; for a detailed discussion, see [6,8].

8 Discussion

We showed how duality properties and geometric considerations are used in studies related to virtual path layouts of chain ATM networks. These dualities follow immediately from the recurrence relations, but a clearer insight was gained with the aid of binary trees (in the shortest path case) and ternary trees (in the general path case). For the general path case we also presented the relation with high dimensional spheres. The duality nature of the solutions, together with the geometric approach, proved to be extremely useful tools in understanding and analyzing the optimal designs. We managed to simplify proofs of known results, derive new results, and improve existing ones.

It might be of interest to further explore such duality relations in various directions. This can be done either for related parameters (such as load measured at vertices, as discussed in [11,9]), or for other topologies (such as trees ([5,11]), meshes ([3,2,11]), or planar graphs ([11])). An interesting direction for extension is suggested for directed networks, following [4]. One might also consult the survey in [18] for a general discussion of these and other extensions.

Of special interest was the use of the high dimensional spheres. The discussion of the use of these spheres and of the applications of this embedding technique suggest this as a promising direction for a further investigation.

References

1. W. Aiello, S. Bhatt, F. Chung, A. Rosenberg, and R. Sitaraman, *Augmented rings networks*, *11th Intl. Conf. on Math. and Computer Modelling and Scientific Computing (ICMCM & SC)* (1997); also: Proceedings of the *6th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Lacanau-Océan, France, 1999, pp. 1-16.
2. L. Becchetti, P. Bertolazzi, C. Gaibisso and G. Gambosi, *On the design of efficient ATM routing schemes*, submitted, 1997.
3. L. Becchetti and C. Gaibisso, *Lower bounds for the virtual path layout problem in ATM networks*, *Proceedings of the 24th Seminar on Theory and Practice of Informatics (SOFSEM)*, Milovny, The Czech Republic, November 1997, pp. 375-382.
4. J-C. Bermond, N. Marlin, D. Peleg and S. Pérennes, *Directed virtual path layout in ATM networks*, *Proceedings of the 12th International Symposium on Distributed Computing (DISC)*, Andros, Greece, September 1998, pp. 75-88.
5. I. Cidon, O. Gerstel and S. Zaks, *A scalable approach to routing in ATM networks*, *8th International Workshop on Distributed Algorithms (WDAG)*, Lecture Notes in Computer Science 857, Springer Verlag, Berlin, 1994, pp.209-222.
6. Y. Dinitz, M. Feighelstein and S. Zaks, *On optimal graph embedded into path and rings, with analysis using l_1 -spheres*, *23th International Workshop on Graph-Theoretic Concepts in Computer Sciences (WG)*, Berlin, Germany, June 1997.
7. T. Eilam, M. Flammini and S. Zaks, *A Complete Characterization of the Path Layout Construction Problem for ATM Networks with Given Hop Count and Load*, *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP)*, Bologna, Italy, pp. 527-537, July 1997.
8. M. Feighelstein, *Virtual path layouts for ATM networks with unbounded stretch factor*, M.Sc. Dissertation, Department of Computer Science, Technion, Haifa, Israel, May 1998.
9. M. Flammini, E. Nardelli and G. Proietti, *ATM layouts with bounded hop count and congestion*, *Proceedings of the 11th International Workshop on Distributed Algorithms (WDAG)*, Saarbrücken, Germany, September 1997, pp. 24-26.
10. M. Feighelstein and S. Zaks, *Duality in chain ATM virtual path layouts*, *Proceedings of the 4th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Monte Verita, Ascona, Switzerland, July 24-26, 1997, pp. 228-239.
11. O. Gerstel, *Virtual Path Design in ATM Networks*, Ph.D. thesis, Department of Computer Science, Technion, Haifa, Israel, December 1995.
12. S. W. Golomb and L. R. Welch, *Perfect Codes in the Lee Metric and the Packing of Polyominoes*, *SIAM Journal on Applied Math.*, vol.18, no.2, January, 1970, pp. 302-317.
13. O. Gerstel, A. Wool and S. Zaks, *Optimal layouts on a chain ATM network*, *Discrete Applied Mathematics*, special issue on Network Communications, 83, 1998, pp. 157-178.
14. O. Gerstel, A. Wool and S. Zaks, *Optimal Average-Case Layouts on Chain Networks*, *Proceedings of the Workshop on Algorithmic Aspects of Communication*, Bologna, Italy, July 11-12, 1997.
15. O. Gerstel and S. Zaks, *The Virtual Path Layout Problem in Fast Networks*, *Proceedings of the 13th ACM Symposium on Principles of Distributed Computing (PODC)*, Los Angeles, CA, U.S.A., August 1994, pp. 235-243.

16. E. Kranakis, D. Krizanc and A. Pelc, *Hop-congestion tradeoffs for ATM networks*, 7th IEEE Symp. on Parallel and Distributed Processing, pp. 662-668.
17. L. Stacho and I. Vrt'o, *Virtual Path Layouts for Some Bounded Degree Networks*. 3rd International Colloquium on Structural Information and Communication Complexity (SIROCCO), Siena, Italy, June 1996.
18. S. Zaks, *Path Layout in ATM Networks*, Proceedings of the 24th Annual Conference on Current Trends in Theory and Practice of Informatics (SOFSEM), Lecture Notes in Computer Science 1338, Springer Verlag, Milovy, The Czech Republic, November 22-29, 1997, pp. 144-160.

On the Lower Bounds for One-Way Quantum Automata^{*}

Farid Ablayev and Aida Gainutdinova

Dept. of Theoretical Cybernetics,
Kazan State University
420008 Kazan, Russia,
{ablayev, aida}@ksu.ru

Abstract. In the paper we consider measured-once (MO-QFA) one-way quantum finite automaton. We prove that for MO-QFA $\mathcal{Q}$ that $(1/2+\varepsilon)$ -accepts ($\varepsilon \in (0, 1/2)$) regular language L it holds that $\dim(\mathcal{Q}) = \Omega\left(\frac{\log \dim(A)}{\log \log \dim(A)}\right)$. In the case $\varepsilon \in (3/8, 1/2)$ we have more precise lower bound $\dim(\mathcal{Q}) = \Omega(\log \dim(A))$ where A is a minimal deterministic finite automaton accepting L , $\dim(\mathcal{Q})$, and $\dim(A)$ are complexity (number of states) of automata $\mathcal{Q}$ and A respectively, $(1/2 - \varepsilon)$ is the error of $\mathcal{Q}$.

The example of language presented in [2] show that our lower bounds are tight enough.

1 Preliminaries

Quantum computation has become a very intensive research area. This is driven by recent results on possibility of constructing effective quantum algorithms for some problems for which effective classical algorithms are unknown. We will mention here only famous quantum algorithm for factoring that operate in polynomial time [8] and refer to the special issue of SIAM J. Comput Vol 26, No 5 which contains seminal articles on quantum computing and analysis of quantum computation models. Many more papers can be found at the Los Alamos preprint server [4].

It is open problem to prove that factoring is hard (or not hard) for classical effective computational models (such as polynomial time Turing machines). But for a known restrictive models (such as finite automata) such effects for some problems were recently proved [2].

Enhanced [5] or Measured-Many [3] Quantum Finite Automata (MM-QFA) is a model of quantum Finite Automata (QFA) where the state of QFA can be measured while each input symbol is processed. It was shown [5] that the class of languages recognized by MM-QFA is proper subset of regular languages. In the

^{*} The research supported by Russia Fund for Basic Research 99-01-00163 and Fund "Russia Universities" 04.01.52. The research partially done while first author visited Aachen and Trier Universities

paper we consider Measure-Once model of one-way quantum finite automaton (MO-QFA) defined in [6] and further investigated in [3]. For this model of QFA the state of the QFA can be measured only when all input symbols are processed. Any language accepted by a MO-QFA can be accepted also by MM-QFA. The converse is not true.

It is appeared that in complexity sense MM-QFA and MO-QFA are incomparable with classic finite (deterministic and probabilistic) automata. That is, as it is proved in [2] MO-QFA can be exponentially smaller (in the sense of number of states) than any probabilistic finite automata with isolated cut point for certain languages. In contrast to this fact it was proved in [7] that for certain language MM-QFAs are exponentially larger than corresponding deterministic finite automata.

In the paper we prove general lower bound for Measured-Once model of QFA. Namely, we prove that for an arbitrary MO-QFA $\mathcal{Q}$ that $(1/2 + \varepsilon)$ -accepts $(\varepsilon \in (0, 1/2))$ regular language L it holds that

$$\dim(\mathcal{Q}) = \Omega\left(\frac{\log \dim(A)}{\log \log \dim(A)}\right).$$

In the case $\varepsilon \in (3/8, 1/2)$ we have more precise lower bound

$$\dim(\mathcal{Q}) \geq \frac{\log \dim(A)}{2 \log(1 + 1/\gamma)}.$$

Here A is a minimal deterministic finite automaton accepting L ; $\dim(\mathcal{Q})$ and $\dim(A)$ are complexity (number of states) of automaton $\mathcal{Q}$ and A respectively, $\gamma = \sqrt{1 + 2\varepsilon - 4\sqrt{1/2 - \varepsilon}}$.

The lower bound proof method use metric property of the space of superpositions of states of bounded error MO-QFA and the metric property of it transition function. The example of language presented in [2] show that our lower bounds are tight enough.

2 Quantum Finite Automata

We consider 1-way quantum finite automata (QFA) model similar to [5], [2] and [3]. 1-way MO-QFA is a tuple

$$\mathcal{Q} = \langle \Sigma, S, \delta, s_0, F \rangle.$$

Here Σ , is an input finite alphabet, S , is a finite set of states (let $d = |S|$), δ is a transition function (we define it below), $s_0 \in S$ is a starting state, and $F \subseteq S$ is a final set of states. Instead of [3] our model of QFA reads inputs from the left to the right which is not important, but somehow more traditionally for automata theory, in [5] also the left-right reading QFA used.

For $s \in S$, $\langle s |$ denote (in Dirac notation) the unit bra-vector (row-vector) with value 1 at s and 0 elsewhere. Consider a set of linear combinations (over the field $\mathbf{C}$ of complex numbers) of these basis vectors

$$\langle \psi | = z_1 \langle s_1 | + z_2 \langle s_2 | + \dots + z_d \langle s_d |. \quad (1)$$

Denote $\mathcal{H}_d$ d -dimensional Hilbert space of such vectors with norm $\|\langle \psi | \|_2 = \sqrt{\sum_{i=1}^d |z_i|^2}$. We will use notation $\|\cdot\|$ for the norm $\|\cdot\|_2$ through the paper. A superposition of states of $\mathcal{Q}$ is any norm 1 element $\langle \psi |$ of $\mathcal{H}_d$. We say that z_i is the amplitude of the state s_i in the superposition $\langle \psi |$ of $\mathcal{Q}$. So, superposition $\langle \psi |$ of states of $\mathcal{Q}$ describes amplitudes to being automaton in a corresponding state. If quantum process of computation of $\mathcal{Q}$ would be terminated then $|z_i|^2$ gives the probability of finding automaton $\mathcal{Q}$ in the state s_i .

The transition function δ maps $S \times \Sigma \times S$ to $\mathbf{C}$. The value $\delta(s, \sigma, s')$ is the amplitude of the state s' in the superposition of states to which $\mathcal{Q}$ goes from the state s after reading σ .

Computation of $\mathcal{Q}$ starts in superposition $\langle s_0 |$. If in the current step of computation a superposition of $\mathcal{Q}$ is

$$\langle \psi | = z_1 \langle s_1 | + z_2 \langle s_2 | + \dots + z_d \langle s_d |$$

then after reading an input $\sigma \in \Sigma$ the new superposition of $\mathcal{Q}$ will be

$$\langle \psi' | = z'_1 \langle s_1 | + z'_2 \langle s_2 | + \dots + z'_d \langle s_d |$$

where $z'_i = \sum_{j=1}^d z_j \delta(s_j, \sigma, s_i)$ for $i \in \{1, 2, \dots, d\}$.

After reading the last symbol of input the computation of $\mathcal{Q}$ terminated and $\langle \psi |$ observed. After that, the superposition of states of $\mathcal{Q}$ collapses to a state $s_i \in S$. This observation gives $s_i \in S$ with the probability $|z_i|^2$. If we get $s_i \in F$, the input is accepted. If $s_i \in S \setminus F$, the input is rejected. The probability of accepting the input is $\sum_{s_i \in F} |z_i|^2$. Clearly we have that the probability of rejecting the input is $\sum_{s_i \notin F} |z_i|^2 = 1 - \sum_{s_i \in F} |z_i|^2$.

Let $\varepsilon \in (0, 1/2)$. We say that QFA $\mathcal{Q}$ $(1/2 + \varepsilon)$ -accepts language $L \subseteq \Sigma^*$ if words from L are accepted with probability at least $1/2 + \varepsilon$ and words from $\Sigma^* \setminus L$ are rejected with probability at least $1/2 + \varepsilon$.

3 Lower Bounds

In this section we present asymptotic lower bounds complexity for bounded error quantum finite automata in terms of complexity of deterministic finite automata and error of quantum computation. Asymptotic complexity characteristics of theorems below can be interpreted as follows. Consider a sequence of regular languages $L_1, \dots, L_n, \dots$ which are presented by minimal deterministic automata $A_1, \dots, A_n, \dots$ with number $\dim(A_1), \dots, \dim(A_n), \dots$ of states with the property $\dim(A_n) \rightarrow \infty$ for $n \rightarrow \infty$. Then for a fix $\varepsilon \in (0, 1/2)$, theorems 1, 2 present lower bounds for quantum automaton for L_n for n large enough.

Theorem 1. *Let $\varepsilon \in (0, 1/2)$. Let $L \subseteq \Sigma^*$ be a language $(1/2 + \varepsilon)$ -accepted by MO-QFA $\mathcal{Q}$. Then it holds that*

$$\dim(\mathcal{Q}) = \Omega\left(\frac{\log \dim(A)}{\log \log \dim(A)}\right)$$

where A is a minimal finite deterministic automaton accepting L .

Theorem below presents more precise lower bound for $\dim(\mathcal{Q})$ for the case $\varepsilon \in (3/8, 1/2)$.

Theorem 2. *Let $\varepsilon \in (3/8, 1/2)$. Let $L \subseteq \Sigma^*$ be a language $(1/2 + \varepsilon)$ -accepted by MO-QFA $\mathcal{Q}$. Then it holds that*

$$\dim(\mathcal{Q}) \geq \frac{\log \dim(A)}{2 \log(1 + 1/\gamma)}$$

where A is a minimal finite deterministic automaton accepting L and $\gamma = \sqrt{1 + 2\varepsilon - 4\sqrt{1/2 - \varepsilon}}$.

The example of language L_p presented in [2] shows that the lower bounds of the theorems 1, 2 are tight enough. For a prime p language L_p over a single letter alphabet defined as follows. $L_p = \{u : |u| \text{ is divisible by } p\}$.

Theorem 3 ([2]). *For any $\varepsilon > 0$, there is a MO-QFA with $O(\log p)$ states recognizing L_p with probability $1 - \varepsilon$.*

Clearly we have that any finite deterministic automaton for L_p needs at least p states. In [2] it was shown that constant bounded error finite probabilistic automata also need at least p number of states to recognize L_p .

4 Proofs

Proofs of theorems 1, 2 use the same idea. We construct a finite deterministic automaton B that recognizes the same language L and

$$\dim(B) \leq \left(1 + \frac{2}{\theta}\right)^{2\dim(\mathcal{Q})}. \quad (2)$$

Proofs of theorems 1, 2 differ only in estimating parameter $\theta > 0$ depending on ε .

4.1 Linear Presentation of QFA

Let $d = \dim(\mathcal{Q})$. In the proof instead of Dirac notation (1) for the bra-vector $\langle \psi |$ of superposition of QFA we will use more brief notation $\psi = (z_1, \dots, z_d)$.

Denote $M(\sigma)$ $d \times d$ complex unitary matrix with rows and columns corresponding to states of $\mathcal{Q}$ where (s, s') -entry of $M(\sigma)$ is $\delta(s, \sigma, s')$. A computation on QFA is a unitary-linear process. That is, computation starts from the initial

vector $\psi(e)$ of amplitudes of states of $\mathcal{Q}$, $e \in \Sigma^*$ denote the empty word. A quantum computation step on $\mathcal{Q}$ (while reading the current input character $\sigma \in \Sigma$) corresponds to multiplying the current vector of amplitudes ψ of states of $\mathcal{Q}$ by transition matrix $M(\sigma)$ to obtain the vector ψ' , $\psi' = \psi M(\sigma)$, representing the amplitude of each states in the next step.

For an input word u denote $\psi(u)$ a superposition of $\mathcal{Q}$ after reading u . That is, after reading an input word $u = \sigma_1, \dots, \sigma_n$ vector $\psi(u)$ is vector of amplitudes of states of $\mathcal{Q}$ where $\psi(u) = \psi(e)M(u) = \psi(e)M(\sigma_1) \cdots M(\sigma_n)$.

From now we will view on $\mathcal{Q}$ as a following deterministic infinite linear automaton

$$\mathcal{LQ} = \langle \Sigma, \Psi, \Delta, \psi(e), \mathcal{F}_\varepsilon \rangle$$

where $\Psi = \{\psi : \psi = \psi(u), u \in \Sigma^*\}$ is the set of states of $\mathcal{LQ}$ (Ψ is a countable subset of $\mathcal{H}_d$), $\Delta : \Psi \times \Sigma \rightarrow \Psi$ is a (linear, one to one) transition function, $\Delta(\psi, \sigma) = \psi M(\sigma) = \psi'$, the initial state of $\mathcal{LQ}$ is $\psi(e)$, and $\mathcal{F}_\varepsilon \subset \Psi$ is the final set of states of $\mathcal{LQ}$. Here we define $\mathcal{F}_\varepsilon$ as

$$\mathcal{F}_\varepsilon = \{\psi : \psi = (z_1, \dots, z_d), \sum_{s_i \in F} |z_i|^2 \geq 1/2 + \varepsilon\}.$$

We define a language L $(1/2 + \varepsilon)$ -accepted by $\mathcal{LQ}$ as

$$L = L(\mathcal{LQ}) = \{u \in \Sigma^* : \Delta(\psi(e), u) \in \mathcal{F}_\varepsilon\}.$$

Selected initial state $\psi(e)$ and the definition of final set $\mathcal{F}_\varepsilon$ provides that $\mathcal{LQ}$ accepts the same language as $\mathcal{Q}$.

4.2 Metric Characterization of QFA

In this subsection we view on linear automaton $\mathcal{LQ}$ as a metric automaton. That is, we investigate metric properties of its set Ψ of states together with metric properties of transition function Δ .

Lemma 1. *Let L be a language $(1/2 + \varepsilon)$ -accepted by $\mathcal{LQ}$. Let $\theta > 0$ and for arbitrary $\psi \in \mathcal{F}_\varepsilon$ and arbitrary $\psi' \notin \mathcal{F}_\varepsilon$ it holds that*

$$\|\psi - \psi'\| \geq \theta.$$

Then there exists a finite deterministic finite automaton B which accepts L and

$$\dim(B) \leq \left(1 + \frac{2}{\theta}\right)^{2d}.$$

Proof: Recall known notions of metric spaces we need in the proof (see for example [1]). Hilbert space $\mathcal{H}_d$ is a metric space with metric defined by the norm $\|\cdot\|$. Points ψ, ψ' from $\mathcal{D}$ are connected through θ -chain if there exists finite set of points $\psi_1, \psi_2, \dots, \psi_m$ from $\mathcal{D}$ such that $\psi_1 = \psi$, $\psi_m = \psi'$ and $\|\psi_i - \psi_{i+1}\| \leq \theta$

for $i \in \{1, \dots, m-1\}$. For metric space its subset $\mathcal{D}$ is called θ -component if arbitrary two points $\psi, \psi' \in \mathcal{D}$ are connected through θ -chain. It is known [1] that if $\mathcal{D}$ is a finite diameter subset of subspace of $\mathcal{H}_d$ (diameter of $\mathcal{D}$ is defined as $\sup_{\psi, \psi' \in \mathcal{D}} \{\|\psi - \psi'\|\}$) then for $\theta > 0$ $\mathcal{D}$ is partitioned to a finite number t of its θ -components. Set Ψ of states of $\mathcal{LQ}$ belongs to sphere of radius 1 which has center $(0, 0, \dots, 0)$ in $\mathcal{H}_d$ because for all $\psi \in \Psi$ it holds that $\|\psi\| = 1$.

From the condition of the lemma it follows that subset $\mathcal{F}_\varepsilon$ of Ψ is a union of some θ -components of Ψ . Transition function Δ preserves the distance. That is, for arbitrary ψ and ξ and arbitrary $\sigma \in \Sigma$ it holds that

$$\|\psi - \xi\| = \|\Delta(\psi, \sigma) - \Delta(\xi, \sigma)\|. \quad (3)$$

From (3) it holds that for $i \in \{1, 2, \dots, t\}$ and for $\sigma \in \Sigma$ there exists $j \in \{1, 2, \dots, t\}$ such that $\Delta(C_i, \sigma) = C_j$. Here $\Delta(C, \sigma)$ defined as $\Delta(C, \sigma) = \bigcup_{\psi \in C} \Delta(\psi, \sigma)$.

Now describe deterministic finite deterministic automaton B which accepts L .

$$B = \langle \Sigma, \mathcal{B}, \delta, C_0, F \rangle$$

where $\mathcal{B} = \{C_1, C_2, \dots, C_t\}$ is a set of states of B ; $\delta : \mathcal{B} \times \Sigma \rightarrow \mathcal{B}$ is a transition function of B such that $\delta(C_i, \sigma) = C_j$ if $\Delta(C_i, \sigma) = C_j$; initial state C_0 is an θ -component of Ψ which contains $\psi(e)$; finite set F of B defined as follows $F = \{C_i : C_i \subseteq \mathcal{F}_\varepsilon\}$.

From the construction of B we have that automaton B and $\mathcal{LQ}$ accept the same language L .

We estimate the number t of θ -components (number of states of B) of Ψ as follows. For each θ -component C select one point $\psi \in C$. If we draw a sphere of the radius $\theta/2$ with the center $\psi \in C$ then all such spheres do not intersect pairwise. All t these spheres are in large sphere of radius $1 + \theta/2$ which has center $(0, 0, \dots, 0)$. The volume of a sphere of a radius r in $\mathcal{H}_d$ is cr^{2d} where the constant c depends on the metric of $\mathcal{H}_d$. Remind that for estimating the volume of the sphere we should take in account that $\mathcal{H}_d$ is d -dimensional *complex* space and each complex point is a 2-dimensional point. So it holds that

$$\dim(B) \leq \frac{c(1 + \theta/2)^{2d}}{c(\theta/2)^{2d}} = \left(1 + \frac{2}{\theta}\right)^{2d}.$$

■

Lemma 2. *Let $\mathcal{LQ}$ $(1/2 + \varepsilon)$ -accepts language L . Then for arbitrary $\psi \in \mathcal{F}_\varepsilon$ and arbitrary $\psi' \notin \mathcal{F}_\varepsilon$ it holds that*

1. $\|\psi - \psi'\| \geq \theta_1 = \varepsilon/\sqrt{d}$
and
2. $\|\psi - \psi'\| \geq \theta_2 = \sqrt{1 + 2\varepsilon - 4\sqrt{1/2 - \varepsilon}}.$

Proof: Let $\psi = (z_1, \dots, z_d)$ and $\psi' = (z'_1, \dots, z'_d)$. Consider norm $\|\cdot\|_1$ defined as $\|\psi\|_1 = \sum_{i=1}^d |z_i|$.

1. From the definition of $\mathcal{LQ}$ it holds that

$$\begin{aligned} 2\varepsilon &\leq \sum_{s_i \in F} (|z_i|^2 - |z'_i|^2) = \sum_{s_i \in F} (|z_i| - |z'_i|)(|z_i| + |z'_i|) \leq \\ &\leq 2 \sum_{s_i \in F} (|z_i| - |z'_i|) \leq 2 \sum_{s_i \in F} |z_i - z'_i| \leq 2\|\psi - \psi'\|_1 \end{aligned}$$

Using inequation

$$a_1 b_1 + a_2 b_2 + \dots + a_d b_d \leq \sqrt{a_1^2 + a_2^2 + \dots + a_d^2} \sqrt{b_1^2 + b_2^2 + \dots + b_d^2}, \quad (4)$$

for $b_1 = b_2 = \dots = b_d = 1$ we get that $\|\psi\|_1 \leq \sqrt{d}\|\psi\|$. Therefore

$$2\varepsilon \leq 2\|\psi - \psi'\|_1 \leq 2\sqrt{d}\|\psi - \psi'\|$$

Finally we have

$$\|\psi - \psi'\| \geq \varepsilon/\sqrt{d}.$$

2. Now consider the next variant of lower bound for $\|\psi - \psi'\|$.

$$\begin{aligned} \|\psi - \psi'\| &= \sqrt{\sum_{i=1}^d |z_i - z'_i|^2} \geq \sqrt{\sum_{i=1}^d (|z_i| - |z'_i|)^2} = \\ &= \sqrt{\sum_{i=1}^d |z_i|^2 + \sum_{i=1}^d |z'_i|^2 - 2 \sum_{i=1}^d |z_i||z'_i|} \geq \\ &\geq \sqrt{\sum_{s_i \in F} |z_i|^2 + \sum_{s_i \notin F} |z'_i|^2 - 2 \sum_{s_i \in F} |z_i||z'_i| - 2 \sum_{s_i \notin F} |z_i||z'_i|}. \end{aligned}$$

From the definition of $\mathcal{LQ}$ we have that $\sum_{s_i \in F} |z_i|^2 \geq 1/2 + \varepsilon$, $\sum_{s_i \notin F} |z'_i|^2 \geq 1/2 + \varepsilon$. Now from the above we get that

$$\|\psi - \psi'\| \geq \sqrt{1/2 + \varepsilon + 1/2 + \varepsilon - 2 \sum_{s_i \in F} |z_i||z'_i| - 2 \sum_{s_i \notin F} |z_i||z'_i|}.$$

Using inequation (4) we get from the above that

$$\|\psi - \psi'\| \geq \sqrt{1 + 2\varepsilon - 2 \sqrt{\sum_{s_i \in F} |z_i|^2} \sqrt{\sum_{s_i \in F} |z'_i|^2} - 2 \sqrt{\sum_{s_i \notin F} |z_i|^2} \sqrt{\sum_{s_i \notin F} |z'_i|^2}}.$$

Using the property $\sum_{s_i \notin F} |z_i|^2 \leq 1/2 - \varepsilon$, $\sum_{s_i \in F} |z'_i|^2 \leq 1/2 - \varepsilon$, $\sum_{s_i} |z_i|^2 \leq 1$, and $\sum_{s_i} |z'_i|^2 \leq 1$, we finally get that

$$\|\psi - \psi'\| \geq \sqrt{1 + 2\varepsilon - 4\sqrt{1/2 - \varepsilon}} = \theta_2.$$

■

Note that the lower bound above for $\|\psi - \psi'\|$ is nontrivial (positive) if $\varepsilon \in (\alpha, 1/2)$ where α is about $3/8$. For $\varepsilon \in (0, \alpha]$ it holds that $1 + 2\varepsilon - 4\sqrt{1/2 - \varepsilon} \leq 0$. In this case the lower bound $\|\psi - \psi'\| \geq \varepsilon/\sqrt{d}$ is more precise.

Now we turn to formal estimating of lower bounds of theorems 1 and 2.

Proof of Theorem 1: From lemma 1 and lemma 2 it holds that

$$t \leq \left(1 + \frac{2\sqrt{d}}{\varepsilon}\right)^{2d}$$

or $\log t = O(d \log d)$. From this we get that

$$d = \Omega\left(\frac{\log t}{\log \log t}\right).$$

Proof of Theorem 2: From lemma 1 and lemma 2 it holds that

$$t \leq \left(1 + \frac{2}{\theta_2}\right)^{2d}$$

or $2d \geq \log t / \log(1 + 2/\theta_2)$. From this we get that

$$d \geq \frac{\log t}{2 \log(1 + 1/\theta_2)}.$$

5 Concluding Remarks

MM-QFA model differ from the MO-QFA model by possibility to measure a configuration of QFA with respect to the three subspaces that correspond to the three possibilities: 1) stop and accept an input, 2) stop and reject the input, 3) continue the computation. We refer to [3] and [5] for formal definitions of MM-QFA model.

Lower bounds similar to that of theorems 1 and 2 are correct for MM-QFA model. These results would be presented in the next paper.

References

1. P. Alexandrov, Introduction to set theory and general topology, Moscow, Nauka, 1977 (in Russian).
2. A. Ambainis and R. Freivalds, 1-way quantum finite automata: strengths, weaknesses and generalization, In *Proceeding of the 39th IEEE Conference on Foundation of Computer Science*, 1998, 332-342. See also quant-ph/9802062 v3,
3. A. Brodsky and N. Pippenger, Characterizations of 1-way quantum finite automata, quant-ph/9903014, 1999
4. <http://xxx.lanl.gov/archive/quant-ph>. See also its Russian mirror: <http://xxx.itep.ru>.
5. A. Kondacs, J. Watrous, On the power of quantum finite state automata. In *Proceeding of the 38th IEEE Conference on Foundation of Computer Science*, 1997, 66-75.
6. C. Moore and J. Crutchfield, Quantum automata and quantum grammars, quant-ph/9707031
7. A. Nayak, Optimal lower bounds for quantum automata and random access codes, *Proceeding of the 40th IEEE Conference on Foundation of Computer Science*, 1999, 369-376. See also quant-ph/9904093
8. P. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, *SIAM J. on Computing*, 26(5), (1997), 1484-1509.

Axiomatizing Fully Complete Models for ML Polymorphic Types^{*}

Samson Abramsky¹ and Marina Lenisa²

¹ Division of Informatics
University of Edinburgh, UK.
`samson@dcs.ed.ac.uk`

² Dipartimento di Matematica e Informatica
Università di Udine, Italy.
`lenisa@dimi.uniud.it`

Abstract. We present *axioms* on models of system F, which are sufficient to show *full completeness* for *ML-polymorphic types*. These axioms are given for *hyperdoctrine* models, which arise as *adjoint models*, i.e. *co-Kleisli categories of linear categories*. Our axiomatization consists of two crucial steps. First, we axiomatize the fact that every relevant morphism in the model generates, under *decomposition*, a possibly *infinite* typed Böhm tree. Then, we introduce an axiom which rules out infinite trees from the model. Finally, we discuss the necessity of the axioms.

Introduction

In this paper we address the problem of *full completeness* (*universality*) for system F. A *categorical* model of a type theory (or logic) is said to be *fully-complete* ([2]) if, for all *types* (*formulae*) A, B , each morphism $f : \llbracket A \rrbracket \rightarrow \llbracket B \rrbracket$, from the interpretation of A into the interpretation of B , is the denotation of a *proof-term* witnessing the entailment $A \vdash B$. The notion of full-completeness is the counterpart of the notion of *full abstraction* for programming languages, in the sense that, if the term language is executable, then a fully-complete model is (up-to a possible quotient) fully-abstract. Besides full completeness, one can ask the question whether the theory induced by a model $\mathcal{M}$ coincides precisely with the syntactical theory or whether more equations are satisfied in $\mathcal{M}$. A model $\mathcal{M}$ is called *faithful* if it realizes exactly the syntactical theory.

The importance of fully (and faithfully) complete, and fully-abstract *denotational* models is that they characterize the space of proofs/programs in a *compositional*, *syntax-independent* way. Recently, Game Semantics has been used to define fully-complete models for various fragments of Linear Logic ([2,8]), and to give fully-abstract models for many programming languages, including PCF [3, 16,20], richer functional languages [19], and languages with non-functional features such as reference types and non-local control constructs [7,17]. Once many

^{*} Work partially supported by Linear FMRX-CT98-0170.

concrete fully-complete and fully-abstract models have been studied, the problem of abstracting and axiomatizing the key properties of these constructions arises naturally. This line of research originated with [1], where axioms, sufficient to prove full abstraction for PCF and full completeness for the simply typed λ -calculus, are given. The axioms for PCF are abstracted from the key lemmas in the proof of full abstraction of the game model of [3]. The axiomatization in [1] makes essential use of the underlying *linear* structure of the game category, and it applies to models of PCF which arise as co-Kleisli categories of some *linear category* [22,11]. These models, given by a linear category and a cartesian closed category, together with a *monoidal adjunction* between the two categories, are called *adjoint models*, following [11,10].

The problem of *full completeness* for second order (polymorphic) λ -calculus, i.e. Girard's system F ([13]), has been extensively studied. In [15], the category of Partial Equivalence Relations (PER) over the open term model of the untyped λ -calculus has been proved to be fully (and faithfully) complete for *algebraic types*, a small subclass of *ML-types*. ML-types are closed types of the form $\forall X_1. \dots \forall X_n. T$, where T is a simple type. A fully-complete model for the whole system F has been provided in [9], but this model is syntactical in nature being defined as a quotient on terms, and therefore it is not sufficiently abstract. More recently, in [14], a fully and faithfully complete game model for system F has been given. But, although this is a game model, it still has a somewhat syntactical flavor—and the construction of the model is extremely complex.

Summarizing the situation, the previous work on the full completeness problem for system F has produced semantically satisfactory models only for algebraic types. In this paper, we present a set of axioms on models of system F, sufficient to guarantee full completeness for ML-types. This axiomatization is put to use in [5,6], in order to provide a concrete denotational model fully-complete for the whole class of ML-types. The axioms presented in this paper are given on the models of system F originated from Lawvere ([18]) which are called *hyperdoctrines* (see also [21]). As in [1], our axiomatization works in the context of *adjoint models* and, although the full completeness result applies to *intuitionistic* types, it makes essential use of the *linear* decomposition of these types. Our axiomatization consists of two crucial steps. First, we axiomatize the fact that every morphism $f : 1 \rightarrow \llbracket T \rrbracket$, where T is an ML-type, generates, under *decomposition*, a possibly *infinite* typed Böhm tree. Then, we introduce an axiom which rules out infinite trees from the model.

The abstract work carried out in this paper has interesting concrete model-theoretic consequences, in that it enables a clean conceptual structure to be given to the proof of full completeness of the concrete model studied in [5,6].

The model construction in [5,6] is based on the technique of *linear realizability*, which is used to define hyperdoctrine adjoint models. This technique consists in constructing a PER category over a *Linear Combinatory Algebra*.

The proof of full completeness of the PER model in [5,6] consists in showing that this model satisfies the axioms presented in this paper. We feel that the

axiomatic technique presented in this paper is both interesting in itself, and illuminates the concrete detailed proof of full completeness of the PER model.

The paper is organized as follows. In Section 1, we recall the syntax ML-types, and we present a result by Statman about theories of the simply typed λ -calculus with *typical ambiguity*. In Section 2, we carry out a *linear* analysis of the notion of $2\lambda\times$ -*hyperdoctrine*, by introducing the notion of *adjoint hyperdoctrine*. In Section 3, we present our set of axioms for full completeness at ML-types. Final remarks and directions for future work appear in Section 4.

The authors are thankful to Furio Honsell, Radha Jagadeesan, Jim Laird, John Longley, Simone Martini and Alex Simpson for useful discussions on some of the issues of the paper.

1 System F and ML Polymorphism

First, we recall the syntax of *ML-types*. Then, we recall a crucial result on the simply typed λ -calculus concerning theories which satisfy *Typical Ambiguity*, namely Statman's *Typical Ambiguity Theorem*. A theory is said to satisfy Typical Ambiguity if two terms are equated if and only if they are equated for all possible substitutions of type variables. Statman's *Typical Ambiguity Theorem* asserts that there is exactly one consistent theory satisfying Typical Ambiguity on the simply typed λ -calculus with infinite type variables: this is the $\beta\eta$ -theory. An immediate consequence of this result is that the only consistent theory on the fragment of system F consisting of *ML-types* is precisely the $\beta\eta$ -theory.

We assume the reader familiar with System F (see e.g. [4]).

We introduce now the class of *ML-polymorphic types*, which correspond to the limited kind of polymorphism allowed in the language ML.

Definition 1 (ML-types). *The class ML-Type of ML-types is defined by:*

$$ML\text{-}Type = \{ \forall \mathbf{X}. T \mid T \in \text{SimType} \wedge FV(T) \subseteq \mathbf{X} \},$$

where $\mathbf{X}$ is an abbreviation for $X_1, \dots, X_n$, for some $n \geq 0$, and *SimType* is the set of simple types over an infinite set of type variables, i.e. the set of types built inductively from the set of type variables only using the arrow type constructor.

Terms of ML-types have essentially the same “combinatorics” as typically ambiguous terms of the simply typed λ -calculus. In fact, any theory on ML-terms induces a theory satisfying Typical Ambiguity.

The following is a result about simply typed λ -calculus with an *infinite* set of type variables first proved in [24].

Theorem 1 (Statman's Typical Ambiguity). *Let T be a type of the simply typed λ -calculus with infinitely many type variables s.t. $FV(T) \subseteq \{X_1, \dots, X_n\}$. If $\not\vdash M =_{\beta\eta} N : T$, then, there exist types $S_1, \dots, S_n$, and $Y \in TVar$, and a term L s.t. $\vdash L[\mathbf{S}/\mathbf{X}] : T[\mathbf{S}/\mathbf{X}] \rightarrow Bool_Y$, where $Bool_Y = Y \rightarrow Y \rightarrow Y$, s.t. $\vdash (LM)[\mathbf{S}/\mathbf{X}] =_{\beta\eta} true : Bool_Y \wedge \vdash (LN)[\mathbf{S}/\mathbf{X}] =_{\beta\eta} false : Bool_Y$, where $true = \lambda x : Y. y : Y.x$ and $false = \lambda x : Y. y : Y.y$.*

Corollary 1. *i) The maximal consistent theory on the simply typed λ -calculus with infinitely many type variables satisfying Typical Ambiguity is the $\beta\eta$ -theory.
ii) The maximal consistent theory on the fragment of system F consisting of ML-types is the $\beta\eta$ -theory.*

Corollary 1ii) implies that any non-trivial fully-complete model for ML-types of system F is necessarily *faithful* at ML-types, i.e. it realizes exactly the $\beta\eta$ -theory at ML-types.

2 Models of System F

We focus on *hyperdoctrine models* of system F. First, we recall the notion of $2\lambda\times$ -hyperdoctrine (see [21]). This essentially corresponds to the notion of *external model* (see [4]). Then, we give the formal definition of full and faithful complete hyperdoctrine model. Finally, we carry out a *linear* analysis of the notion of $2\lambda\times$ -hyperdoctrine. This will allow us to express conditions which guarantee full completeness of the model w.r.t. ML-types. In particular, we introduce a categorical notion of *adjoint hyperdoctrine*. Adjoint hyperdoctrines arise as *co-Kleisli* indexed categories of *linear* indexed categories.

In what follows, we assume that all indexed categories which we consider are strict (see e.g. [4,12] for more details on indexed categories).

Definition 2 (Hyperdoctrine, [18,21]). *A $2\lambda\times$ -hyperdoctrine is a triple $(\mathcal{C}, \mathbf{G}, \forall)$, where:*

- $\mathcal{C}$ is the base category, it has finite products, and it consists of a distinguished object $\mathcal{U}$ which generates all other objects using the product operation $\times$. We will denote by $\mathcal{U}^m$, for $m \geq 0$, the objects of $\mathcal{C}$.
- $\mathbf{G} : \mathcal{C}^{op} \rightarrow \mathbf{CCCat}$ is a $\mathcal{C}$ -indexed cartesian closed category such that: for all $\mathcal{U}^m$, the underlying collection of objects of the cartesian closed fibre category $\mathbf{G}(\mathcal{U}^m)$ is indexed by the collection of morphisms from $\mathcal{U}^m$ to $\mathcal{U}$ in $\mathcal{C}$, i.e. the objects of $\mathbf{G}(\mathcal{U}^m)$ are the morphisms in $\text{Hom}_{\mathcal{C}}(\mathcal{U}^m, \mathcal{U})$, and, for any morphism $f : \mathcal{U}^m \rightarrow \mathcal{U}^n$ in $\mathcal{C}^{op}$, the cartesian closed functor $\mathbf{G}(f) : \mathbf{G}(\mathcal{U}^n) \rightarrow \mathbf{G}(\mathcal{U}^m)$, called *reindexing functor* and denoted by f^* , is such that, for any object $h : \mathcal{U}^n \rightarrow \mathcal{U}$, $f^*(h) = f; h$;
- For each object $\mathcal{U}^m$ of $\mathcal{C}$, there are functors $\forall_m : \mathbf{G}(\mathcal{U}^m \times \mathcal{U}) \rightarrow \mathbf{G}(\mathcal{U}^m)$ s.t.
 - $\forall_m$ is right adjoint to the functor $\pi_m^* : \mathbf{G}(\mathcal{U}^m) \rightarrow \mathbf{G}(\mathcal{U}^m \times \mathcal{U})$, where $\pi_m : \mathcal{U}^m \times \mathcal{U} \rightarrow \mathcal{U}^m$ is the projection in $\mathcal{C}$;
 - $\forall_m$ satisfies the Beck-Chevalley condition.

Any $2\lambda\times$ -hyperdoctrine can be endowed with a notion of interpretation $\llbracket \cdot \rrbracket$ for the language of system F. Types with free variables in $X_1, \dots, X_m$ are interpreted by morphisms from $\mathcal{U}^m$ to $\mathcal{U}$ in $\mathcal{C}$, i.e. by objects of $\mathbf{G}(\mathcal{U}^m)$: $\llbracket X_1, \dots, X_m \vdash T \rrbracket : \mathcal{U}^m \rightarrow \mathcal{U}$. Well-typed terms, i.e. $X_1, \dots, X_m; x_1 : T_1, \dots, x_n : T_n \vdash M : T$, are interpreted by morphisms in the category $\mathbf{G}(\mathcal{U}^m)$:

$\llbracket X_1, \dots, X_m; x_1 : T_1, \dots, x_n : T_n \vdash M : T \rrbracket : \llbracket X \vdash T_1 \rrbracket \times \dots \times \llbracket X \vdash T_n \rrbracket \rightarrow \llbracket X \vdash T \rrbracket$.
See e.g. [12] for more details.

Definition 3 (Full and Faithful Completeness). Let $\mathcal{M} = (\mathcal{C}, \mathbf{G}, \forall, \llbracket \cdot \rrbracket)$ be a $2\lambda\times$ -hyperdoctrine. $\mathcal{M}$ is fully and faithfully complete w.r.t. the class of closed types $\mathcal{T}$ if, for all $T \in \mathcal{T}$,

$$\forall f \in \text{Hom}_{\mathbf{G}_{(1)}}(1, \llbracket T \rrbracket). \exists (!)\beta\eta\text{-normal form } M. \vdash M : T \wedge f = \llbracket \vdash M : T \rrbracket.$$

Before presenting the notion of adjoint hyperdoctrine, we recall some definitions:

Definition 4 (Linear Category, [22,11]). A linear category is a symmetric monoidal closed category $(\mathcal{L}, I, \otimes, \multimap)$ with

- a symmetric monoidal comonad $(!, \text{der}, \delta, \phi, \phi')$ on $\mathcal{L}$;
- monoidal natural transformations with components $\text{weak}_A : !A \rightarrow I$ and $\text{con}_A : !A \rightarrow !A \otimes !A$ such that
 - each $(!A, \text{weak}_A, \text{con}_A)$ is a commutative comonoid,
 - weak_A and con_A are $!$ -coalgebra maps from $(!A, \delta_A)$ to (I, ϕ'_I) , and from $(!A, \delta_A)$ to $(!A \otimes !A, \delta_A \otimes \delta_A; \phi_{!A, !A})$, respectively.
 - all coalgebra maps between free $!$ -coalgebras preserve the canonical structure.

Definition 5 (Adjoint Model, [10]). An adjoint model is specified by

1. a symmetric monoidal closed category $(\mathcal{L}, I, \otimes, \multimap)$;
2. a cartesian closed category $(\mathcal{C}, 1, \times, \rightarrow)$;
3. a symmetric monoidal adjunction from $\mathcal{C}$ to $\mathcal{L}$.

Now we give the *indexed* version of the notion of adjoint model:

Definition 6 (Indexed Adjoint Model). An indexed adjoint model is specified by

1. a symmetric monoidal closed indexed category $\mathbf{L} : \mathcal{C}^{op} \rightarrow \text{SMCCat}$;
2. a cartesian closed indexed category $\mathbf{G} : \mathcal{C}^{op} \rightarrow \text{CCCat}$;
3. a symmetric monoidal indexed adjunction from $\mathbf{G}$ to $\mathbf{L}$.

In the following definition, which is inspired by [23], we capture those hyperdoctrines which arise from a co-Kleisli construction over an *indexed linear category*.

Definition 7 (Adjoint Hyperdoctrine).

An adjoint hyperdoctrine is a quadruple $(\mathcal{C}, \mathbf{L}, \mathbf{G}, \forall)$, where:

- $\mathcal{C}$ is the base category, it has finite products, and it consists of a distinguished object $\mathcal{U}$ which generates all other objects using the product operation $\times$. We will denote by $\mathcal{U}^m$, for $m \geq 0$, the objects of $\mathcal{C}$.
- $\mathbf{L} : \mathcal{C}^{op} \rightarrow \text{LCat}$ is a $\mathcal{C}$ -indexed linear category such that: for all $\mathcal{U}^m$, the underlying collection of objects of the linear fibre category $\mathbf{L}(\mathcal{U}^m)$ is indexed by the collection of morphisms from $\mathcal{U}^m$ to $\mathcal{U}$ in $\mathcal{C}$.
- $\mathbf{G} : \mathcal{C}^{op} \rightarrow \text{CCCat}$ is the $\mathcal{C}$ -indexed cartesian closed co-Kleisli category of $\mathbf{L}$.
- For each object $\mathcal{U}^m$ of $\mathcal{C}$, there are functors $\forall_m : \mathbf{G}(\mathcal{U}^m \times \mathcal{U}) \rightarrow \mathbf{G}(\mathcal{U}^m)$ s.t.
 - $\forall_m : \mathbf{G}(\mathcal{U}^m \times \mathcal{U}) \rightarrow \mathbf{G}(\mathcal{U}^m)$ is right adjoint to the functor $\mathbf{G}(\pi_m) : \mathbf{G}(\mathcal{U}^m) \rightarrow \mathbf{G}(\mathcal{U}^m \times \mathcal{U})$, where $\pi_m : \mathcal{U}^m \times \mathcal{U} \rightarrow \mathcal{U}^m$ is the projection in $\mathcal{C}$;
 - $\forall_m : \mathbf{G}(\mathcal{U}^m \times \mathcal{U}) \rightarrow \mathbf{G}(\mathcal{U}^m)$ satisfies the Beck-Chevalley condition.

An adjoint hyperdoctrine is, in particular, an indexed adjoint model, and it gives rise to a $2\lambda\times$ -hyperdoctrine:

Theorem 2. *Let $(\mathcal{C}, \mathbf{L}, \mathbf{G}, \forall)$ be an adjoint hyperdoctrine. Then*

- i) the categories $\mathbf{L}$ and $\mathbf{G}$ form an indexed adjoint model;*
- ii) $(\mathcal{C}, \mathbf{G}, \forall)$ is an hyperdoctrine.*

Remark. In the definition of adjoint hyperdoctrine, we require the indexed categories $\mathbf{L}$ and $\mathbf{G}$ to form an adjoint model, but we assume the existence of a family of functors $\forall_m$ only on the fibre categories of $\mathbf{G}$. Therefore, we have a model of linear *first order* types, but not of linear *higher order* types, and our definition does not capture models of L/NL system F, i.e. system F with both linear and intuitionistic types. But our notion of model is sufficient for dealing with ML-types, and for expressing axioms for full completeness at ML-types.

3 Axiomatizing Models Fully Complete for ML Types

We isolate sufficient conditions on adjoint hyperdoctrine models for system F, in order to guarantee full completeness at ML-polymorphic types. These conditions amount to the six axioms of Subsection 3.1. Our axiomatization of full completeness for ML polymorphism is in the line of the work in [1], where an axiomatic approach to full abstraction/full completeness for PCF/simply typed λ -calculus is presented. These axiomatizations are inspired by the proof of full abstraction of the Game Semantics model for PCF of [3]. Our axiomatization of full completeness for ML-types consists of two parts:

1) Axioms for ensuring the Decomposition Theorem. This theorem allows to recover the *top-level structure* of the (possibly infinite) Böhm tree denoted by morphisms from the terminal object into the interpretation of an ML-type in the fibre category $\mathbf{G}(1)$. The axioms for the Decomposition Theorem (Axioms 1–5 of Section 3.1) make essential use of the linear category underlying an adjoint hyperdoctrine. These axioms (apart from the axioms 1 and 3), are expressed by requiring some canonical maps between suitable spaces of morphisms in the fibre categories $\mathbf{L}(\mathcal{U})$ to be isomorphisms.

2) A *Finiteness Axiom*, which allows to rule out infinite Böhm trees from the model.

Notice that, by definition of interpretation function on a hyperdoctrine (see e.g. [12]), morphisms f in $\mathbf{G}(1)$ from the terminal object of $\mathbf{G}(1)$ into $\llbracket \vdash T \rrbracket$, where $\forall \mathbf{X}. T_1 \rightarrow \dots \rightarrow T_n \rightarrow X_k$ is an ML-type, are λ -definable if and only if morphisms of $\mathbf{G}(\mathcal{U})$ from $\times_{i=1}^n \llbracket \mathbf{X} \vdash T_i \rrbracket$ into $\llbracket \mathbf{X} \vdash X_k \rrbracket$ are λ -definable.

Namely, $f = \llbracket \vdash \Lambda \mathbf{X}. \lambda \mathbf{x} : \mathbf{T}. x_i M_1 \dots M_{q_i} : \forall \mathbf{X}. \mathbf{T} \rightarrow X_k \rrbracket$ if and only if $\Lambda^{-1}(\hat{f}) = \llbracket \mathbf{X}; \mathbf{x} : \mathbf{T} \vdash x_i M_1 \dots M_{q_i} : X_k \rrbracket$, where $\hat{}$ denotes the inverse of the bijection given by the adjunction between $\forall$ and π^* in Definition 2. Therefore, from now on we focus on the space of morphisms of $\mathbf{G}(\mathcal{U})$ from $\times_{i=1}^n \llbracket \mathbf{X} \vdash T_i \rrbracket$ into $\llbracket \mathbf{X} \vdash X_k \rrbracket$, where $T_1, \dots, T_n$ are simple types.

We start by presenting the main result of this section, i.e. the Decomposition Theorem. The proof of this theorem follows from the Strong Decomposition Theorem 4, which is proved in Subsection 3.1.

If a morphism f of $\mathbf{G}(\mathcal{U})$ from $\times_{i=1}^n \llbracket \mathbf{X} \vdash T_i \rrbracket$ into $\llbracket \mathbf{X} \vdash X_k \rrbracket$ is λ -definable, then $f = \llbracket \mathbf{X}; \mathbf{x} : \mathbf{T} \vdash x_i M_1 \dots M_{q_i} : X_k \rrbracket$, for some $\mathbf{X}; \mathbf{x} : \mathbf{T} \vdash M_1 : U_{i1}, \dots, \mathbf{X}; \mathbf{x} : \mathbf{T} \vdash M_{q_i} : U_{iq_i}$. I.e., making evident the *top-level* structure of the Böhm tree:

$$f = \llbracket \mathbf{X}; \mathbf{x} : \mathbf{T} \vdash x_i : T_i \rrbracket \bullet \llbracket \mathbf{X}; \mathbf{x} : \mathbf{T} \vdash M_1 : U_{i1} \rrbracket \bullet \dots \bullet \llbracket \mathbf{X}; \mathbf{x} : \mathbf{T} \vdash M_{q_i} : U_{iq_i} \rrbracket ,$$

where $\bullet$ denotes the application in the model.

The Decomposition Theorem allows recovering the top-level structure of the Böhm tree corresponding to f in the following sense:

Theorem 3 (Decomposition). *Let $(\mathcal{C}, \mathbf{L}, \mathbf{G}, \forall)$ be an adjoint hyperdoctrine satisfying Axioms 1–5 of Section 3.1. Let $T = T_1 \rightarrow \dots \rightarrow T_n \rightarrow X_k$ be a simple type with $FV(T) \subseteq \{X_1, \dots, X_n\}$, where, for all $i = 1, \dots, n$, $T_i = U_{i1} \rightarrow \dots \rightarrow U_{iq_i} \rightarrow X_i$. Then, for all $f \in \text{Hom}_{\mathbf{G}(\mathcal{U})}(\times_{i=1}^n \llbracket \mathbf{X} \vdash T_i \rrbracket, \llbracket \mathbf{X} \vdash X_k \rrbracket)$, there exist $i \in \{1, \dots, n\}$ and $g_j \in \text{Hom}_{\mathbf{G}(\mathcal{U})}(\times_{i=1}^n \llbracket \mathbf{X} \vdash T_i \rrbracket, \llbracket \mathbf{X} \vdash U_{ij} \rrbracket)$, for all $j = 1, \dots, q_i$, such that $f = \llbracket \mathbf{X}; \mathbf{x} : \mathbf{T} \vdash x_i : T_i \rrbracket \bullet g_1 \dots \bullet g_{q_i}$.*

Since the g 's appearing in the Decomposition Theorem still live (up-to-uncurrying) in a space of morphisms denoting a simple type, we can keep on iterating the decomposition, expanding in turn these g 's, thus getting a possibly infinite tree from f . If the Decomposition Theorem holds, in order to get the full completeness result, we are left only to rule out morphisms generating trees whose height is infinite, which would correspond to infinite typed Böhm trees. This is expressed in the *Finiteness Axiom* 6 below.

3.1 The Axioms

The first axiom expresses the fact that the type $\forall \mathbf{X}. X_k$ is empty.

Axiom 1 (Base) $\text{Hom}_{\mathbf{L}(\mathcal{U})}(1, \pi_k) = \emptyset$,
 where 1 is the terminal object in $\mathbf{G}(\mathcal{U})$, and $\pi_k : \mathcal{U} \rightarrow \mathcal{U}$ denotes the k -th projection in $\mathbf{G}(\mathcal{U})$, i.e. $\pi_k = \text{weak}_1 \otimes \dots \otimes \text{weak}_{k-1} \otimes \text{der}_k \otimes \text{weak}_{k+1} \otimes \dots \otimes \text{weak}_n$.

The following axiom allows extracting *one* copy of the type of the head variable, corresponding to the *first* use of this variable. The property expressed by this axiom is *truly* linear. In fact, in order to state it, we are implicitly using the canonical morphism $!A \rightarrow A \otimes !A$ to capture the idea of a “first occurrence”.

Axiom 2 (Linearization of Head Occurrence)

$\text{case}_i \{ \sigma_i \}_{i=1, \dots, n} : \coprod_{i=1}^n \text{Hom}_{\mathbf{L}(\mathcal{U})}(h_i, (\mathbf{h} \multimap \pi_k)) \simeq \text{Hom}_{\mathbf{L}(\mathcal{U})}(\mathbf{h}, \pi_k)$,
 where

- $\coprod$ denotes coproduct in Set ;
- $\mathbf{h} = \otimes_{i=1}^n !h_i$ and $\forall i \in \{1, \dots, n\}$. $h_i = \otimes_{j=1}^{q_i} !l_{ij} \multimap \pi_{p_i}$;
- $\sigma_i : \text{Hom}_{\mathbf{L}(\mathcal{U})}(h_i, (\mathbf{h} \multimap \pi_k)) \rightarrow \text{Hom}_{\mathbf{L}(\mathcal{U})}(\mathbf{h}, \pi_k)$ is the following canonical morphism: $\Lambda^{-1}; \text{Hom}_{\mathbf{L}(\mathcal{U})}(\pi_i; \tau, id_{\pi_k}); \text{Hom}_{\mathbf{L}(\mathcal{U})}(\text{con}_{\mathbf{h}}, id_{\pi_k})$, where $\tau : I \otimes \dots \otimes I \otimes h_i \otimes I \dots \otimes I \simeq h_i$.

The following axiom reflects a form of coherence of the type of the head variable w.r.t. the global type of the term. I.e., if $\mathbf{T} \rightarrow X_k$ is the type of a term, then the type of the head variable must be of the shape $\mathbf{U} \rightarrow X_i$, with $k = i$.

Axiom 3 (Type Coherence) $\text{Hom}_{\mathbf{L}(\mathbf{U})}(l \multimap \pi_i, h \multimap \pi_k) = \emptyset$, if $i \neq k$.

The following axiom expresses the fact that the only thing that we can do with a *linear* functional parameter is applying it to an argument *which does not itself depend on the parameter*. Note that, again, linearity is essential here. For example, if copying were allowed, then the argument could itself contain further occurrences of the parameter.

Axiom 4 (Linear Function Extensionality)

$$\text{Hom}_{\mathbf{L}(\mathbf{U})}((\cdot), \text{id}_{\pi_k}) : \text{Hom}_{\mathbf{L}(\mathbf{U})}(h, l) \simeq \text{Hom}_{\mathbf{L}(\mathbf{U})}(l \multimap \pi_k, h \multimap \pi_k) .$$

The following axiom expresses the fact that morphisms from $!f$ to $!g$ in the fibre category $\mathbf{L}(\mathbf{U})$ have *uniform* behavior in all threads.

Axiom 5 (Uniformity of Threads)

$$\tau_1 : \text{Hom}_{\mathbf{L}(\mathbf{U})}(!h, !l) \simeq \text{Hom}_{\mathbf{L}(\mathbf{U})}(!h, l) : \tau_2 ,$$

where $\tau_1 = \text{Hom}_{\mathbf{L}(\mathbf{U})}(\text{id}_{!h}, \text{der}_l)$, $\tau_2 = \lambda f \in \text{Hom}_{\mathbf{L}(\mathbf{U})}(!h, l). (f)_{h,l}^\dagger$, and $(\cdot)_{h,l}^\dagger : \text{Hom}_{\mathbf{L}(\mathbf{U})}(!h, l) \rightarrow \text{Hom}_{\mathbf{L}(\mathbf{U})}(!h, !l)$ is the canonical morphism given by the comonad $!$.

The final axiom in our axiomatization guarantees that the tree generated via repeated applications of the Decomposition Theorem 4 to morphisms in $\text{Hom}_{\mathbf{L}(\mathbf{U})}(\otimes_{i=1}^n !h_i, \pi_k)$, where $h_i = \otimes_{j=1}^{q_i} !l_{ij} \multimap \pi_{p_i}$ is *finite*.

Axiom 6 (Finiteness) *There exists a size function*

$\mathcal{H} : \bigcup \{ \text{Hom}_{\mathbf{L}(\mathbf{U})}(\otimes_{i=1}^n !h_i, \pi_k) \mid k \in \mathbf{N}, h_i = \otimes_{j=1}^{q_i} !l_{ij} \multimap \pi_{p_i} \} \longrightarrow \mathbf{N}$,
such that $\forall j \in \{1, \dots, q_i\}$. $\mathcal{H}(g_j) < \mathcal{H}(f)$, where the g_j 's are defined in the Decomposition Theorem 4.

Notice that Axioms 1–5 actually give a stronger form of decomposition than Theorem 3. Namely, the decomposition is *unique*, and it holds for all morphisms in $\text{Hom}_{\mathbf{L}(\mathbf{U})}(\mathbf{h}, \pi_k)$, where $\mathbf{h} = \otimes_{i=1}^n !h_i$ and, for all $i = 1, \dots, n$, $h_i = \otimes_{j=1}^{q_i} !l_{ij} \multimap \pi_{p_i}$, and not just for the morphisms which are definable.

Theorem 4 (Strong Decomposition). *Let $(\mathcal{C}, \mathbf{G}, \mathbf{L}, \forall)$ be an adjoint hyperdoctrine satisfying Axioms 1–5 of Section 3.1. Let $f \in \text{Hom}_{\mathbf{L}(\mathbf{U})}(\mathbf{h}, \pi_k)$, where $\mathbf{h} = \otimes_{i=1}^n !h_i$ and, for all $i = 1, \dots, n$, $h_i = \otimes_{j=1}^{q_i} !l_{ij} \multimap \pi_{p_i}$. Then there exist a unique i and unique $g_1, \dots, g_{q_i}$ such that, for all $j = 1, \dots, q_i$, $g_j \in \text{Hom}_{\mathbf{L}(\mathbf{U})}(\mathbf{h}, l_{ij})$, and $f = \text{con}_{\mathbf{h}}; (\pi_k \otimes \langle g_1, \dots, g_{q_i} \rangle^\dagger); \text{Ap}$.*

Proof. If $\mathbf{h} = 1$, then, by Axiom 1, we have immediately the thesis. Otherwise, by Axiom 2, there exists a unique $i \in \{1, \dots, n\}$ and a unique $f' \in \text{Hom}_{\mathbf{L}(\mathcal{U})}(h_i, \mathbf{h} \multimap \pi_k)$ such that $f = \text{con}_{\mathbf{h}}; \pi_i \otimes \text{id}_{\mathbf{h}}; \Lambda^{-1}(f')$. By Axiom 3, $\pi_i = \pi_k$. By Axiom 4, there exists a unique $g \in \text{Hom}(\mathbf{h}, \mathbf{l}_i)$, where $\mathbf{l}_i = \otimes_{j=1}^{q_i} !l_{ij}$, such that $f' = g \multimap \pi_k = \Lambda((\text{id} \otimes g); \text{Ap})$. Then $f = \text{con}_{\mathbf{h}}; \pi_k \otimes g; \text{Ap}$. Finally, by Axiom 5, and by the universal property of the product, we obtain $g = \langle g_1, \dots, g_{q_i} \rangle^\dagger$. $\square$

Finally, we can show the main result:

Theorem 5 (Axiomatic Full Completeness). *Let $\mathcal{M}$ be an adjoint hyperdoctrine. If $\mathcal{M}$ satisfies Axioms 1–6, then $\mathcal{M}$ is fully and faithfully complete at ML-types.*

Proof. Let $\forall \mathbf{X}.T = T_1 \rightarrow \dots \rightarrow T_n \rightarrow X_k$ be an ML-type, and let $f \in \text{Hom}_{\mathbf{L}(1)}(!I, \llbracket \forall \mathbf{X}.T \rrbracket)$. Using the Decomposition Theorem, one can easily prove, by induction on the measure provided by Axiom 6 that there exists $\mathbf{X}; \mathbf{x} : T \vdash x_i M_1 \dots M_{q_i} : X_k$ such that $\overline{\Lambda(f)} = \llbracket \mathbf{X}; \mathbf{x} : T \vdash x_i M_1 \dots M_{q_i} : X_k \rrbracket$, where $\overline{}$ is the bijection given by the adjunction between $\forall$ and π^* in Definition 2. Then $f = \llbracket \Lambda \mathbf{X}. \lambda \mathbf{x} : T. x_i M_1 \dots M_{q_i} : \forall \mathbf{X}.T \rightarrow X_k \rrbracket$. $\square$

4 Final Remarks and Directions for Future Work

As we remarked earlier, the Strong Decomposition Theorem 4 is stronger than the Decomposition Theorem 3 in two respects. First of all, it provides a decomposition for *all* morphisms of a certain shape, and not just for those whose domains and codomains are *denotations* of types. Secondly, it guarantees the *uniqueness* of the decomposition. Correspondingly, the axioms could be weakened either by considering only spaces of morphisms whose domains and codomains are *denotations* of types, instead of generic objects of appropriate “top-level” shape, or by substituting the isomorphisms requirements by weaker conditions, which only ensure the existence of a decomposition. More precisely, the first kind of restriction is obtained by taking, e.g. in the *Linearization of Head Occurrence*, $\mathbf{h}$ to be $\otimes_{i=1}^n !\llbracket T_i \rrbracket$, where each T_i is a simple type with free variables in $\mathbf{X}$. In order to guarantee the existence of a decomposition, it is sufficient to ask that the canonical morphisms in Axioms 2 and 4 and the morphism $\lambda f \in \text{Hom}(!\mathbf{h}, l).(f)^\dagger$ in Axiom 5 are *surjective* maps. By weakening the axioms in either or both of these ways, we still get a set of sufficient conditions for full completeness. Notice that, if we take the weaker form of the axioms which imply only the existence of a decomposition, we need indeed Statman’s result to conclude faithfulness.

In the concrete model of PERs over the LCA of partial involutions of [5], we succeeded in proving the weak variant of the axioms obtained by restricting the spaces of morphisms. But we conjecture that the full strong form holds.

Finally, notice that all the Axioms 1–6 in the strong form presented in Section 3 are *consistent*, since they are satisfied in the underlying category of the adjoint hyperdoctrine induced by the linear term model. Moreover, Axiom 1 is

trivially *necessary*. The question of the necessity of the Axioms 2–6 in their weak or strong form remains open.

In this paper, we have presented axioms for full completeness at ML-types. A natural question arises: what happens beyond ML-types. Here is a partial answer. Already at the type $\text{Nat} \rightarrow \text{Nat}$, where *Nat* is the type of Church's numerals, i.e. $\forall X.(X \rightarrow X) \rightarrow X \rightarrow X$, the PER model over the linear term combinatory algebra is not fully-complete. In fact, all recursive functions can be encoded in the type $\text{Nat} \rightarrow \text{Nat}$. A similar problem arises also if we consider the LCA of partial involutions studied in [5]. PER models as they are defined in [5], do not seem to give full-completeness beyond ML-types. In principle, one could give axioms for full completeness w.r.t. larger fragments of system F, but, at the moment, the real challenge is that of isolating a fragment of system F which properly includes the ML-types, while still admitting “good” fully-complete models.

References

1. S.Abramsky. Axioms for Definability and Full Completeness, in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, G. Plotkin, C. Stirling and M. Tofte, eds., MIT Press, 2000, 55–75.
2. S.Abramsky, R.Jagadeesan. Games and Full Completeness for Multiplicative Linear Logic, *J. of Symbolic Logic* **59**(2), 1994, 543–574.
3. S.Abramsky, R.Jagadeesan, P.Malacaria. Full Abstraction for PCF, 1996, *Inf. and Comp.*, to appear.
4. A.Asperti, G.Longo. *Categories, Types ad Structures*, Foundations of Computing Series, The MIT Press, 1991.
5. S.Abramsky, M.Lenisa. Fully Complete Models for ML Polymorphic Types, Report **ECS-LFCS-99-414**, University of Edinburgh, October 1999.
6. S.Abramsky, M.Lenisa. A Fully Complete PER Model for ML Polymorphic Types, *CSL'2000 Conf. Proc.*, to appear.
7. S.Abramsky, G.McCusker. Full abstraction for idealized Algol with passive expressions, *TCS* **227**, 1999, 3–42.
8. S.Abramsky, P.Mellies. *Concurrent Games and Full Completeness*, *LICS'99*.
9. V.Breazu-Tannen, T.Coquand. Extensional models for polymorphism, *TCS* **59**, 1988, 85–114.
10. N.Benton, P.Wadler. Linear Logic, Monads and the Lambda Calculus, *LICS'96*.
11. G.Bierman. What is a categorical Model of Intuitionistic Linear Logic?, *TLCA '95 Conf. Proc.*, LNCS, 1995.
12. R.Crole, *Categories for Types*, Cambridge University Press, 1993.
13. J.Y.Girard. Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur, Thèse d'Etat, Université Paris VII, 1972.
14. D.J.D.Hughes. *Hypergame Semantics: Full Completeness for System F*. D.Phil. thesis, University of Oxford, submitted 1999.
15. J.Hyland, E.Robinson, G.Rosolini. Algebraic types in PER models, *MFPS Conf. Proc.*, M.Main *et al.* eds, LNCS **442**, 1990, 333–350.
16. M.Hyland, L.Ong. On full abstraction for PCF, *Inf. and Comp.*, 1996, to appear.
17. J.Laird. Full abstraction for functional languages with control, *LICS'97*.
18. F.Lawvere. Equality in hyperdoctrines and the comprehension schema as an adjoint functor, *Proc. Symp. on Applications of Categorical Logic*, 1970.

19. G.McCusker. Games and full abstraction for FPC, *LICS'96*.
20. H.Nickau. Hereditarily sequential functionals, Proc. of the Symposium *Logical Foundations for Computer Science*, LNCS **813**, 1994.
21. A.Pitts. Polymorphism is set-theoretic constructively, *CTCS'88* Conf. Proc., D.Pitt ed., LNCS **283**, 1988.
22. R.Seely. Linear logic, *-autonomous categories and cofree coalgebras, in *Category theory, computer science and logic*, American Math. Society, 1987.
23. R.Seely. Polymorphic linear logic and topos models, Math. Reports, Academy of Science (Canada) XII, 1990.
24. Statman. λ -definable functionals and $\beta\eta$ -conversion, *Arch. Math. Logik* **23**, 1983.

Measure Theoretic Completeness Notions for the Exponential Time Classes

Klaus Ambos-Spies

Mathematisches Institut, Universität Heidelberg,
Im Neuenheimer Feld 294, D-69120 Heidelberg, Germany
ambos@math.uni-heidelberg.de

Abstract. The resource-bounded measure theory of Lutz leads to variants of the classical hardness and completeness notions. While a set A is hard (under polynomial time many-one reducibility) for a complexity class $\mathcal{C}$ if every set in $\mathcal{C}$ can be reduced to A , a set A is *almost hard* if the class of reducible sets has measure 1 in $\mathcal{C}$, and a set A is *weakly hard* if the class of reducible sets does not have measure 0 in $\mathcal{C}$. If, in addition, A is a member of $\mathcal{C}$ then A is *almost complete* and *weakly complete* for $\mathcal{C}$, respectively. Weak hardness for the exponential time classes $\mathbf{E} = \mathbf{DTIME}(2^{\text{lin}(n)})$ and $\mathbf{EXP} = \mathbf{DTIME}(2^{\text{poly}(n)})$ has been extensively studied in the literature, whereas the nontriviality of the concept of almost completeness has been established only recently.

Here we continue the investigation of these measure theoretic hardness notions for the exponential time classes and we establish the relations among these notions which had been left open. In particular, we show that almost hardness for $\mathbf{E}$ and $\mathbf{EXP}$ are independent. Moreover, there is a set in $\mathbf{E}$ which is almost complete for $\mathbf{EXP}$ but not weakly complete for $\mathbf{E}$. These results exhibit a surprising degree of independence of the measure concepts for $\mathbf{E}$ and $\mathbf{EXP}$.

Finally, we give structural separations for some of these concepts and we show the nontriviality of almost hardness for the bounded query reducibilities of fixed norm.

1 Introduction

Lutz [12] introduced measure concepts for sufficiently closed complexity classes containing the exponential time class $\mathbf{E} = \mathbf{DTIME}(2^{\text{lin}(n)})$. These measure concepts are natural resource-bounded variants of the classical Lebesgue measure which are based on the characterization of measure-0 classes in terms of certain betting games, called martingales. Lutz and others have used these resource-bounded measure notions for the analysis of quantitative aspects of the structure of complexity classes. These investigations focussed on the exponential time classes $\mathbf{E}$ and $\mathbf{EXP} = \mathbf{DTIME}(2^{\text{poly}(n)})$. See the surveys of Lutz [14] and Ambos-Spies and Mayordomo [1] for details.

The measure theoretic analysis of complexity classes led to natural measure theoretic counterparts to the classical completeness and hardness notions. While

a set H is *hard* (under some given polynomial time reducibility) for a complexity class C if all sets in this class can be reduced to H , a set is *almost hard* for C if the class of reducible sets has measure 1 in C , i.e., intuitively, if the problems in C which cannot be reduced can be neglected in the sense of measure; and a set is *weakly hard* if the class of reducible sets does not have measure 0 in C , i.e., the class of reducible sets cannot be neglected in terms of measure. A set is *complete* (*almost complete*, *weakly complete*) for C if the set is a member of C and hard (almost hard, weakly hard) for C .

Lutz [13] has shown that for the most commonly used reducibility, namely the polynomial time many-one (p-m for short) reducibility, completeness and weak completeness differ for both **E** and **EXP**. In fact, Ambos-Spies, Terwijn and Zheng [5] have shown that the class of weakly complete sets in **E** (**EXP**) has measure 1 in **E** (**EXP**), whereas, by a result of Mayordomo [15], the class of complete sets has measure 0 in **E** (**EXP**). So a typical set in **E** (**EXP**) is weakly complete but not complete. The nontriviality of the concept of almost completeness for **E** and **EXP** was established only recently by Ambos-Spies, Merkle, Reimann and Terwijn [3] who have shown that there is a set in **E** which is almost complete but not complete for **E** and **EXP**. By previous results of Regan, Sivakumar and Cai [16] and Ambos-Spies, Neis and Terwijn [4], however, the almost complete sets are as rare as the complete sets from the point of measure, namely have measure 0 in **E** and **EXP**.

The classes **E** and **EXP** are closely related to each other, whence it is interesting to compare the various hardness notions for these two classes with each other. In case of the measure theoretic hardness notions this contributes to a better understanding of the relations among the underlying measure concepts for **E** and **EXP**.

Since the class **EXP** is the closure of the smaller class **E** under p-m-equivalence, classical hardness for **E** and **EXP** coincide, whence, for a set in **E**, completeness for **E** and **EXP** is the same. In contrast to this, Juedes and Lutz [11] have shown that weak hardness for **E** implies weak hardness for **EXP** but not vice versa. The question about the relation between the almost hardness notions for the two exponential time classes raised in [3] had been open.

Here we show that almost hardness for **E** and **EXP** are independent: We construct sets A_1 and A_2 in **E** such that A_1 is almost complete for **E** but not almost complete for **EXP** while A_2 is almost complete for **EXP** but not almost complete for **E**, in fact not even weakly complete for **E**. In the light of the previous results on weak hardness this strong independence might be surprising and it indicates that the measure systems for **E** and **EXP** might be less related to each other than one might have expected.

Our new results together with the results in the literature give a complete characterization of the valid implications among the completeness and hardness notions introduced above. By further showing that combining the properties of weak hardness for **E** and almost hardness for **EXP** does not suffice to force almost hardness for **E**, we can extend this to a complete characterization of the possible relations of any arity.

In the second part of the paper we continue the investigation of some other aspects of almost hardness and completeness: First, we give structural separations of completeness and almost completeness in terms of immunity, and of almost completeness and weak completeness in terms of incompressibility. Second, we look at almost completeness for the exponential time classes under bounded truth-table and Turing reductions of fixed norm, and we show the nontriviality of these concepts.

The outline of the paper is as follows. In Sect. 2 we introduce some basic concepts of and results on Lutz's resource-bounded measure theory needed in the following. In Sect. 3 we present our results on the relation between almost completeness for **E** and **EXP**. The structural separations are given in Sect. 4, and almost completeness under the bounded query reducibilities is discussed in Sect. 5. Due to lack of space, proofs are omitted. The proofs of our main results (Theorems 13, 14, 15, 19 and 22) refine the technique of [3] and some of them are quite technical.

Our notation is standard. Unexplained notation can be found in [1] and [3].

2 Resource-Bounded Measure

In this section we shortly introduce the fragment of Lutz's resource-bounded measure theory which we will need in the following. For a more comprehensive presentation and intuitive explanations we refer to the surveys by Lutz [14] and Ambos-Spies and Mayordomo [1]. Our presentation follows the latter. For fixed time bounds the measure defined there slightly differs from Lutz's original definition but it leads to the same measure on **E** and **EXP**.

Definition 1. *A betting strategy s is a function $s : \{0, 1\}^* \rightarrow [0, 1]$. The (normed) martingale $d_s : \{0, 1\}^* \rightarrow [0, \infty)$ induced by a betting strategy s is inductively defined by $d_s(\lambda) = 1$ and $d_s(xi) = 2 \cdot |i - s(x)| \cdot d_s(x)$ for $x \in \{0, 1\}^*$ and $i \in \{0, 1\}$. A martingale is a martingale induced by some strategy. A martingale d succeeds on a set A if*

$$\limsup_{n \rightarrow \infty} d(A \upharpoonright n) = \infty,$$

and d succeeds on a class $\mathcal{C}$ if d succeeds on every member A of $\mathcal{C}$.

A class $\mathcal{C}$ has (classical) Lebesgue measure 0, $\mu(\mathcal{C}) = 0$, iff some martingale succeeds on $\mathcal{C}$. By imposing resource bounds, the martingale concept is used for defining resource-bounded measure concepts.

Definition 2. *Let $t : \mathbb{N} \rightarrow \mathbb{N}$ be a recursive function. A $t(n)$ -martingale d is a martingale induced by a rational valued betting strategy s such that $s(x)$ can be computed in $t(|x|)$ steps for all strings x . A class $\mathcal{C}$ has $t(n)$ -measure 0, $\mu_{t(n)}(\mathcal{C}) = 0$, if some $t(n)$ -martingale succeeds on $\mathcal{C}$; and $\mathcal{C}$ has $t(n)$ -measure 1, $\mu_{t(n)}(\mathcal{C}) = 1$, if the complement $\overline{\mathcal{C}}$ has $t(n)$ -measure 0.*

For the definition of a measure on a general deterministic time class one works with corresponding time-bounded martingales. For defining measures on the exponential time classes $\mathbf{E} = \mathbf{DTIME}(2^{\text{lin}(n)})$ and $\mathbf{EXP} = \mathbf{DTIME}(2^{\text{poly}(n)})$ polynomial martingales and quasi polynomial martingales are adequate.

Definition 3. (i) A p-martingale d is a $q(n)$ -martingale for some polynomial q . A class $\mathbf{C}$ has p-measure 0, $\mu_p(\mathbf{C}) = 0$, if some p-martingale succeeds on $\mathbf{C}$, and $\mu_p(\mathbf{C}) = 1$ if $\mu_p(\overline{\mathbf{C}}) = 0$. A class $\mathbf{C}$ has measure 0 in $\mathbf{E}$, $\mu(\mathbf{C}|\mathbf{E}) = 0$, if $\mu_p(\mathbf{C} \cap \mathbf{E}) = 0$ and $\mathbf{C}$ has measure 1 in $\mathbf{E}$, $\mu(\mathbf{C}|\mathbf{E}) = 1$, if $\mu(\overline{\mathbf{C}}|\mathbf{E}) = 0$.

(ii) A p_2 -martingale d is a $2^{(\log n)^k}$ -martingale for some $k \geq 1$. A class $\mathbf{C}$ has p_2 -measure 0, $\mu_{p_2}(\mathbf{C}) = 0$, if some p_2 -martingale succeeds on $\mathbf{C}$, and $\mu_{p_2}(\mathbf{C}) = 1$ if $\mu_{p_2}(\overline{\mathbf{C}}) = 0$. A class $\mathbf{C}$ has measure 0 in $\mathbf{EXP}$, $\mu(\mathbf{C}|\mathbf{EXP}) = 0$, if $\mu_{p_2}(\mathbf{C} \cap \mathbf{EXP}) = 0$ and $\mathbf{C}$ has measure 1 in $\mathbf{EXP}$, $\mu(\mathbf{C}|\mathbf{EXP}) = 1$, if $\mu(\overline{\mathbf{C}}|\mathbf{EXP}) = 0$.

Lutz [12] has shown that the measures on $\mathbf{E}$ and $\mathbf{EXP}$ are consistent. In particular, $\mathbf{E}$ itself does not have measure 0 in $\mathbf{E}$ whereas every "slice" $\mathbf{DTIME}(2^{kn})$ of $\mathbf{E}$ ($k \geq 1$) has measure 0 in $\mathbf{E}$ (and, similarly, for $\mathbf{EXP}$ and $\mathbf{DTIME}(2^{n^k})$). Here we will use the characterization of the measures on $\mathbf{E}$ and $\mathbf{EXP}$ in terms of resource-bounded random sets.

Definition 4. A set R is $t(n)$ -random if no $t(n)$ -martingale succeeds on R .

Note that, for t and t' such that $t(n) \leq t'(n)$ a.e., any $t'(n)$ -random set is $t(n)$ -random too.

Lemma 5. For any class $\mathbf{C}$,

- (i) $\mu(\mathbf{C}|\mathbf{E}) = 0$ iff there is a number k such that $\mathbf{C} \cap \mathbf{E}$ does not contain any n^k -random set.
- (ii) $\mu(\mathbf{C}|\mathbf{EXP}) = 0$ iff there is a number k such that $\mathbf{C} \cap \mathbf{EXP}$ does not contain any $2^{(\log n)^k}$ -random set.

3 Weak and Almost Completeness for $\mathbf{E}$ and $\mathbf{EXP}$

Next we will introduce the completeness and hardness notions for the exponential time classes studied in this paper. The underlying polynomial time reducibility will be the p-m-reducibility unless explicitly stated otherwise. We let $P_{\leq}(A) = \{B : B \leq_m^p A\}$ denote the lower span of A . Then a set A is *hard* for a class $\mathbf{C}$ if every set in $\mathbf{C}$ can be reduced to A , i.e., if $\mathbf{C}$ is contained in $P_{\leq}(A)$. If, in addition, A is in $\mathbf{C}$ then A is *complete* for $\mathbf{C}$. The measure theoretic hardness notions are obtained by relaxing the requirement that $\mathbf{C}$ is entirely contained in $P_{\leq}(A)$.

Definition 6 (Lutz [13], Zheng). A set A is *almost hard* for a class $\mathbf{C}$ if $P_{\leq}(A)$ has measure 1 in $\mathbf{C}$, and A is *weakly hard* for $\mathbf{C}$ if $P_{\leq}(A)$ does not have measure 0 in $\mathbf{C}$. If, in addition, $A \in \mathbf{C}$ then A is *almost complete* for $\mathbf{C}$ and *weakly complete* for $\mathbf{C}$, respectively.

Intuitively, if A is almost hard for C then the part of C which cannot be reduced to A can be neglected while if A is weakly hard for C a nonnegligible part of C can be reduced to A (but the nonreducible part might be nonnegligible too). So, for $C \in \{\mathbf{E}, \mathbf{EXP}\}$, it is immediate by definition and by consistency of the measure on C that

$$\begin{aligned} A \text{ hard (complete) for } C &\Rightarrow A \text{ almost hard (complete) for } C \\ &\Rightarrow A \text{ weakly hard (complete) for } C \end{aligned} \quad (1)$$

In fact it has been shown that these implications are strict. The first important step towards these separations was done by Lutz [13] who has shown that there are weakly complete sets for $C \in \{\mathbf{E}, \mathbf{EXP}\}$ which are not complete for $\mathbf{E}$. Then a characterization of weak completeness in terms of randomness provided a stronger separation.

Theorem 7 (Ambos-Spies, Terwijn, Zheng [5]). *A set A is weakly hard for $\mathbf{E}$ ($\mathbf{EXP}$) iff there is an n^2 -random set in $P_{\leq}(A) \cap \mathbf{E}$ ($P_{\leq}(A) \cap \mathbf{EXP}$).*

In particular this shows that any n^2 -random set in $\mathbf{E}$ ($\mathbf{EXP}$) is weakly complete for $\mathbf{E}$ ($\mathbf{EXP}$). Since, on the other hand, no n^2 -random set is p-btt-complete for $\mathbf{E}$ or $\mathbf{EXP}$ (see [4]) this implies the following result.

Corollary 8 ([5]). *There is a weakly complete set for $\mathbf{E}$ ($\mathbf{EXP}$) which is not p-btt-complete for $\mathbf{E}$ ($\mathbf{EXP}$).*

Together with an earlier observation of Regan, Sivakumar and Cai this gives the separation of weak hardness and almost hardness.

Lemma 9 (Regan, Sivakumar, Cai [16]). *Let C be a class which is closed under union and intersection (or under symmetric difference) and which has measure 1 in $\mathbf{E}$ ($\mathbf{EXP}$). Then $\mathbf{E}$ ($\mathbf{EXP}$) is contained in C . Hence, if A is almost hard for $\mathbf{E}$ ($\mathbf{EXP}$) then A is p-btt-hard for $\mathbf{E}$ ($\mathbf{EXP}$).*

The separation of hardness and almost hardness was obtained only recently.

Theorem 10 (Ambos-Spies, Merkle, Reimann, Terwijn [3]). *There is a set in $\mathbf{E}$ which is almost complete for $\mathbf{E}$ and $\mathbf{EXP}$ but is complete neither for $\mathbf{E}$ nor for $\mathbf{EXP}$.*

Other work in the literature is directed to the question how the corresponding hardness notions for the two exponential time classes are related to each other. Since $\mathbf{EXP}$ is the closure of $\mathbf{E}$ under p-m-equivalence, hardness for $\mathbf{E}$ and $\mathbf{EXP}$ coincides.

Proposition 11 (see [9]). *For any set $A \in \mathbf{EXP}$ there is a set $A' \in \mathbf{DTIME}(2^n)$ which is p-m-equivalent to A . Hence, hardness for $\mathbf{E}$ and $\mathbf{EXP}$ coincides, and, for any set $A \in \mathbf{E}$, A is complete for $\mathbf{E}$ iff A is complete for $\mathbf{EXP}$.*

Juedes and Lutz have shown, however, that if we pass from hardness to weak hardness the coincidence of the notions for **E** and **EXP** is lost in part.

Theorem 12 (Juedes and Lutz [11]). *Every weakly hard set for **E** is weakly hard for **EXP** too; but there is a set in **E** which is weakly complete for **EXP** but not weakly hard for **E**.*

Examples of weakly complete sets for **EXP** which are not weakly hard for **E** are the $2^{(\log n)^2}$ -random sets in **EXP**.

In [3], where the nontriviality of the almost hardness notions was shown, the question of the relation between almost hardness for **E** and almost hardness for **EXP** was left open. Here we show that – in contrast to the above results – almost hardness for **E** and **EXP** is completely independent. Moreover, almost hardness for **EXP** in general does not even imply weak hardness for **E**, whereas, by (1) and by Theorem 12, every almost hard set for **E** is weakly hard for **EXP**.

Theorem 13. *There is a set A in **E** which is almost complete for **EXP** but not almost complete for **E**, in fact not even weakly complete for **E**.*

Theorem 14. *There is a set A in **E** which is almost complete for **E** but not almost complete for **EXP**.*

Together with the above cited results from the literature these theorems show that the implications shown in Fig. 1 are the only implications which hold among the hardness notions for **E** and **EXP**. Moreover, there it does not matter whether we consider arbitrary sets A or only sets $A \in \mathbf{E}$. In order to completely

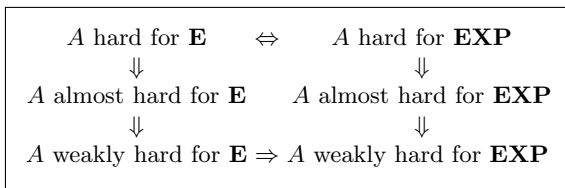


Fig. 1.

classify the inclusion structure of the above hardness classes we need one more fact, namely, that almost hardness for **EXP** combined with weak hardness for **E** in general does not imply almost hardness for **E**.

Theorem 15. *There is a set A in **E** which is almost complete for **EXP** and weakly complete for **E** but not almost complete for **E**.*

Now, if, for $C \in \{\mathbf{E}, \mathbf{EXP}\}$, we let

$$\mathcal{H} = \{A : A \text{ hard for } \mathbf{E}\} = \{A : A \text{ hard for } \mathbf{EXP}\}$$

$$\mathcal{AH}(C) = \{A : A \text{ almost hard for } C\}$$

$$\mathcal{WH}(C) = \{A : A \text{ weakly hard for } C\}$$

then we can summarize the relations among these classes in the following theorem.

Theorem 16 (Main Classification Theorem).

$$\mathcal{H} \subsetneq \mathcal{AH}(\mathbf{E}) \subsetneq \mathcal{WH}(\mathbf{E}) \subsetneq \mathcal{WH}(\mathbf{EXP}) \quad (2)$$

$$\mathcal{H} \subsetneq \mathcal{AH}(\mathbf{EXP}) \subsetneq \mathcal{WH}(\mathbf{EXP}) \quad (3)$$

$$\begin{aligned} &\text{The class } \mathcal{AH}(\mathbf{EXP}) \text{ properly splits the classes} \\ &\mathcal{AH}(\mathbf{E}) - \mathcal{H}, \mathcal{WH}(\mathbf{E}) - \mathcal{AH}(\mathbf{E}) \text{ and } \mathcal{WH}(\mathbf{EXP}) - \mathcal{WH}(\mathbf{E}). \end{aligned} \quad (4)$$

Moreover (2) - (4) remain valid if we intersect all classes with $\mathbf{E}$.

Theorem 16 is illustrated in Fig. 2 where all regions are nonempty and the shaded region describes the location of the class $\mathcal{AH}(\mathbf{EXP})$ within the other classes.

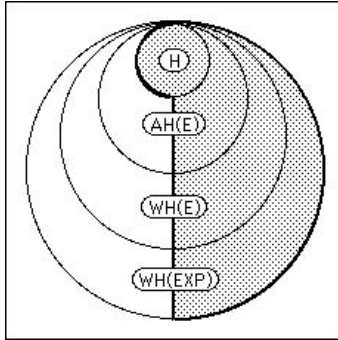


Fig. 2.

Proof. Claims (2) and (3) are immediate by the results from the literature cited above: For $C \in \{\mathbf{E}, \mathbf{EXP}\}$, $\mathcal{H} \subsetneq \mathcal{AH}(C)$ holds by Theorem 10 and $\mathcal{AH}(C) \subsetneq \mathcal{WH}(C)$ holds by Corollary 8 and Lemma 9. Finally, $\mathcal{WH}(\mathbf{E}) \subsetneq \mathcal{WH}(\mathbf{EXP})$ holds by Theorem 12.

For the proof of (4) it suffices to establish the existence of sets $A_n \in \mathbf{E}$ ($n \leq 5$) such that $A_0, A_1 \in \mathcal{AH}(\mathbf{E}) - \mathcal{H}$, $A_2, A_3 \in \mathcal{WH}(\mathbf{E}) - \mathcal{AH}(\mathbf{E})$, $A_4, A_5 \in \mathcal{WH}(\mathbf{EXP}) - \mathcal{WH}(\mathbf{E})$, and the sets with even index are in $\mathcal{AH}(\mathbf{EXP})$ whereas the sets with odd index are not in $\mathcal{AH}(\mathbf{EXP})$.

The existence of the required sets A_1, A_2 and A_4 follows from our new Theorems 14, 15 and 13, respectively. The existence of A_0 follows from Theorem 10, the existence of A_3 from Corollary 8 and Lemma 9. Finally, for the existence of A_5 , note that, by the remarks following Theorems 12 and 7, for every $2^{(\log n)^2}$ -random set $R \in \mathbf{EXP}$, $R \in \mathcal{WH}(\mathbf{EXP}) - \mathcal{WH}(\mathbf{E})$ and R is not p-btt-complete for $\mathbf{EXP}$, whence $R \notin \mathcal{AH}(\mathbf{EXP})$. So, since the hardness notions are invariant under p-m-equivalence, it suffices to let A_5 be a set in $\mathbf{E}$ which is p-m-equivalent to R . $\square$

4 Structural Separations

Structural separations of classical completeness notions under various reducibilities are discussed in Buhrmann and Torenvliet [8] and in Homer [9]. The results in the literature on the measure theoretic completeness notions (for $\mathbf{E}$ and $\mathbf{EXP}$) easily imply structural separations of completeness and weak completeness in terms of (bi-)immunity and incompressibility. Here we refine these results by giving structural separations of completeness from almost completeness in terms of (bi-)immunity, and of almost completeness from weak completeness in terms of incompressibility.

Recall that a set A is **P-immune** if A is infinite and A does not contain any infinite set $B \in \mathbf{P}$ as a subset, and A is **P-bi-immune** if A and its complement $\bar{A}$ are **P-immune**. A strengthening of **P-bi-immunity** is incompressibility under p-m-reductions: A set A is p-m-incompressible if for any p-m-reduction f from A to any set B , f is almost one-to-one. Here we will also consider the somewhat stronger incompressibility under p-btt(1)-reductions: A set A is p-btt(1)-incompressible if for any p-btt(1)-reduction (α, h) from A to any set B the selector function h is almost one-to-one. Note that

$$\begin{aligned} A \text{ p-btt(1)-incompressible} &\Rightarrow A \text{ p-m-incompressible} \\ &\Rightarrow A \text{ P-bi-immune} \end{aligned} \tag{5}$$

where the first implication is immediate by definition and the second implication is straightforward (see e.g. [6]). Moreover, the implications are strict. E.g., for p-btt(1)-incompressible A , $(A \cap \overline{\{0\}^*}) \cup \{0^{2n} : 0^n \in A\} \cup \{0^{2n+1} : 0^n \notin A\}$ is p-m-incompressible but not p-btt(1)-incompressible and, for p-m-incompressible A , $A \oplus A$ is **P-bi-immune** but not p-m-incompressible.

Berman [7] has shown that no complete set for $\mathbf{E}$ ($\mathbf{EXP}$) is **P-immune**. On the other hand, all n^2 -random sets are **P-bi-immune** (Mayordomo [15]). In fact, as Juedes and Lutz [10] have shown, n^2 -random sets are p-m-incompressible, and the proof can be easily modified to show the following.

Lemma 17. *Every n^2 -random set is p-btt(1)-incompressible.*

By Theorems 7 and 12 the above implies the following structural separations.

Theorem 18. *While no complete set for $\mathbf{E}$ or $\mathbf{EXP}$ is **P-(bi-)immune** there are weakly complete sets for $\mathbf{E}$ and $\mathbf{EXP}$ which are **P-(bi-)immune**, p-m-incompressible and p-btt(1)-incompressible.*

By refining the proof of Theorem 10 we can show that there are also almost complete sets for **E** and **EXP** which are **P**-bi-immune thereby giving a structural separation of completeness and almost completeness.

Theorem 19. *There is a **P**-bi-immune set A in **E** which is almost complete for **E** and **EXP**.*

On the other hand we can show, however, that almost complete sets for **E** and **EXP** cannot be $p\text{-btt}(1)$ -incompressible. By Theorem 18 this gives a structural separation for almost completeness and weak completeness.

Theorem 20. *If A is almost complete for **E** or **EXP** then A is $p\text{-btt}(1)$ -compressible.*

We do not know whether in Theorem 20 $p\text{-btt}(1)$ -compressibility can be replaced by the more common $p\text{-m}$ -compressibility.

5 Almost Completeness under Bounded Query Reducibilities

By Lemma 9, almost completeness and completeness (for **E** and **EXP**) coincide for polynomial time Turing ($p\text{-T}$), truth-table ($p\text{-tt}$) and bounded truth-table ($p\text{-btt}$) reducibility. Moreover, Ambos-Spies et al. [3] have shown that almost completeness for **E** (**EXP**) coincides with (the apparently stronger) almost completeness under $p\text{-1-li}$ -reducibility, i.e., one-to-one and length increasing reductions, and (the apparently weaker) $p\text{-btt}(1)$ -reducibility, which parallels previous results on completeness and weak completeness (see [1] for details).

Here we show the nontriviality of almost hardness for the bounded query reducibilities of constant norm $c \geq 1$, namely, for the nonadaptive bounded truth-table reducibilities $\leq_{\text{btt}(c)}^P$ and for the adaptive bounded Turing reducibilities $\leq_{\text{bT}(c)}^P$.

By Lemma 9, any almost $p\text{-btt}(c)$ - or $p\text{-bT}(c)$ -hard set is $p\text{-btt}$ -hard, i.e., $p\text{-btt}(c')$ -hard for some constant c' . By generalizing an observation on almost $p\text{-m}$ -hard sets in [1] we obtain the following bound on the size of c' .

Lemma 21. *Let $C \in \{\mathbf{E}, \mathbf{EXP}\}$ and $c \geq 1$. If A is almost $p\text{-btt}(c)$ -hard (almost $p\text{-bT}(c)$ -hard) for C then A is $p\text{-btt}(2c)$ -hard ($p\text{-bT}(2c)$ -hard) for C .*

We separate almost hardness from hardness for the bounded query reducibilities by showing that the bound obtained in the preceding lemma is optimal. This also yields a hierarchy theorem for almost hardness when we let the norm of the underlying reducibility grow.

Theorem 22. *For every $c \geq 1$ there is a set $A \in \mathbf{E}$ such that A is almost $p\text{-btt}(c)$ -complete for **E** and **EXP** but A is not $p\text{-btt}(2c-1)$ -hard for **E**, in fact not even $p\text{-bT}(2c-1)$ -hard for **E**.*

Corollary 23. *For $C \in \{\mathbf{E}, \mathbf{EXP}\}$ and for every number $c \geq 1$ there is an almost $\text{p-btt}(c)$ -complete set for C which is not $\text{p-btt}(c)$ -complete for C and there is an almost $\text{p-btt}(c+1)$ -complete set for C which is not almost $\text{p-btt}(c)$ -complete for C (and similarly for p-bT in place of p-btt).*

By further refinements of the results of Sect. 3 and by using some results from [2] we can obtain a complete characterization of the (binary) relations among the various hardness notions for the exponential time classes under the bounded query reducibilities.

Acknowledgements. We thank an anonymous referee of [3] for suggesting the investigation of almost hardness for the bounded query reducibilities.

References

1. K. Ambos-Spies and E. Mayordomo. Resource-bounded measure and randomness. In: A. Sorbi (ed.), *Complexity, logic, and recursion theory*, p.1-47, Dekker, 1997.
2. K. Ambos-Spies, E. Mayordomo and X. Zheng. A comparison of weak completeness notions. In: *Proceedings of the 11th Ann. IEEE Conference on Computational Complexity*, p.171–178, IEEE Computer Society Press, 1996.
3. K. Ambos-Spies, W. Merkle, J. Reimann, and S.A. Terwijn. Almost complete sets. In: *STACS 2000*, LNCS 1770, p.419–430, Springer, 2000.
4. K. Ambos-Spies, H.-C. Neis and S.A. Terwijn. Genericity and measure for exponential time. *Theoretical Computer Science*, 168:3–19, 1996.
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. J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity*, volume I. Springer, 1995.
7. L. Berman. *Polynomial reducibilities and complete sets*. Ph.D. thesis, Cornell University, 1977.
8. H. Buhrmann and L. Torenvliet. On the structure of complete sets. In: *Proceedings of the 9th Ann. Structure in Complexity Conference*, p.118–133, IEEE Computer Society Press, 1994.
9. S. Homer. Structural properties for complete problems for exponential time. In: *Complexity Theory Retrospective II* (Hemaspaandra, L.A. et al., eds.), p.135–153, Springer, 1997.
10. D.W. Juedes and J.H. Lutz. The complexity and distribution of hard problems. *SIAM Journal on Computing*, 24:279–295, 1995.
11. D.W. Juedes and J.H. Lutz. Weak completeness in $\mathbf{E}$ and $\mathbf{E}_2$. *Theoretical Computer Science*, 143:149–158, 1995.
12. J.H. Lutz. Almost everywhere high nonuniform complexity. *Journal of Computer and System Sciences*, 44:220–258, 1992.
13. J.H. Lutz. Weakly hard problems. *SIAM Journal on Computing* 24:1170–1189, 1995.
14. J.H. Lutz. The quantitative structure of exponential time. In: *Complexity Theory Retrospective II* (Hemaspaandra, L.A. et al., eds.), p.225–260, Springer, 1997.
15. E. Mayordomo. Almost every set in exponential time is $\mathbf{P}$ -bi-immune. *Theoretical Computer Science*, 136:487–506, 1994.
16. K. Regan, D. Sivakumar and J.-Y. Cai. Pseudorandom generators, measure theory and natural proofs. In: *Proceedings of the 36th Ann. IEEE Symposium on Foundations of Computer Science*, p.171–178, IEEE Computer Society Press, 1995.

Edge-Bisection of Chordal Rings [★]

Lali Barrière and Josep Fàbrega

Dept. de Matemàtica Aplicada i Telemàtica
Universitat Politècnica de Catalunya, Barcelona
`lali,matjfc@mat.upc.es`

Abstract. An edge-bisector of a graph is a set of edges whose removing separates the graph into two subgraphs of same order, within one. The edge-bisection of a graph is the cardinality of the smallest edge-bisector. The main purpose of this paper is to estimate the quality of general bounds on the edge-bisection of Cayley graphs. For this purpose we have focused on chordal rings of degree 3. These graphs are Cayley graphs on the dihedral group and can be considered as the simplest Cayley graphs on a non-abelian group (the dihedral group is metabelian). Moreover, the natural plane tessellation used to represent and manipulate these graphs can be generalized to other types of tessellations including abelian Cayley graphs. We have improved previous bounds on the edge-bisection of chordal rings and we have shown that, for any fixed chord, our upper bound on the edge-bisection of chordal rings is optimal up to an $O(\log n)$ factor. Finally, we have given tight bounds for optimal chordal rings, that are those with the maximum number of vertices for a given diameter.

1 Introduction

An edge-bisector of a graph is a set of edges whose removing separates the graph into two subgraphs of same order, within one. The edge-bisection of a graph is the cardinality of the smallest edge-bisector.

The edge-bisection is a significant parameter for the study of graphs as models for interconnection networks. Indeed, on one hand, the physical constraints that apply to the design of networks include limits on the edge-bisection, node size and channel width [1]. On the other hand, the edge-bisection is an important factor for determining the complexity of algorithms in which information has to be exchanged between two halves of the network. For instance, algorithms based on the divide-and-conquer strategy perform better on graphs that have large edge-bisections [17].

A vertex-bisector of a graph of order n is a set of at most $\frac{n}{3}$ vertices whose removing separates the graph into two subgraphs of same order. The vertex-bisection of a graph is the cardinality of the smallest vertex-bisector. If k , b , and d denote respectively the edge-bisection, the vertex-bisection and the maximum degree of a graph, then $b \leq k \leq d \cdot b$. This implies that, although the two

[★] Work supported by the Spanish Research Council (Comisión Interministerial de Ciencia y Tecnología, CICYT) under project TIC-97-0963.

problems are not equivalent, the edge-bisection and the vertex-bisection have the same order for graphs of bounded degree.

Computing the edge-bisection or the vertex-bisection of arbitrary graphs yields NP-complete decision problems [13]. However, many algorithms and heuristics have been designed, though often for restricted classes of graphs. They are of very different nature: improvement techniques [16], maxflow-mincut theorem [9], algebraic optimization [8], simulated annealing [19], etc. Despite these efforts, good approximation algorithms are known for dense graphs only [4], and exact algorithms are known for the special cases of trees and bounded-width planar graphs only [9]. Actually, even for the case of planar graphs, which have vertex-bisection $O(\sqrt{n})$ (see [18]), no approximation algorithms are known.

Besides the look for efficient algorithms, other works focus on the exact computation of the edge-bisection of specific families of graphs, as hypercubes and tori, among others [17]. More generally, many works were devoted to the search of tight lower and/or upper bounds on the edge-bisection, as a special case of graph partitioning (for instance [11]). In order to reduce the difficulty of the edge-bisection and vertex-bisection computation, many attention was devoted to graphs with symmetry properties. In particular, thanks to the properties of the derived series of a group, upper bounds on the order of vertex-bisectors of Cayley graphs on solvable groups are given in [2,7].

The main goal of this paper is to estimate the quality of the bounds of these two latter papers. For this purpose we will focus on chordal rings [3]. These graphs are Cayley graphs on the dihedral group. They can be obtained from a cycle of even length $2n$ by adding a chord to every even vertex in a regular manner, that is for every i , node $2i$ is connected by the chord to the odd node $2i + d$. The reason why we are interested in these graphs is two folded. On one hand, they can be considered as the simplest Cayley graphs on a non-abelian group. On the other hand, the natural plane tessellation used to represent and manipulate these graphs can be generalized to other types of tessellations including abelian Cayley graphs.

The dihedral group D_n is the set of the plane symmetries of a regular n -gon. As said before, D_n is not abelian. However, it is metabelian [21]. Recall that a metabelian group is a solvable non-abelian group with derived length 2. That is, it is a non-abelian group Γ whose set of commutators (i.e., the set $\{aba^{-1}b^{-1}, a \in \Gamma, b \in \Gamma\}$) generates an abelian subgroup. The order of D_n is $2n$, and the set of commutators of D_n is a subgroup isomorphic to $\mathbb{Z}_n$, if n is odd, and to $\mathbb{Z}_{n/2}$, if n is even. From the results derived in [2], we get that, up to a factor of 3, the edge-bisection of a chordal ring of order $2n$ is at most

$$(6!)^3 \cdot 2n \cdot n^{-1/(f(6)+1)} \text{ if } n \text{ is odd,} \quad (12!)^3 \cdot 2n \cdot \left(\frac{n}{2}\right)^{-1/(f(12)+1)} \text{ if } n \text{ is even,}$$

where $f(r) = \prod_{i=2}^r (\log_2 i + 2)$. Note that, for this bound to be smaller than the order of the graph, this order has to be greater than 10^{100} .

Contributions. In this paper we improve the bounds of [2] on the edge-bisection of chordal rings. Roughly speaking, we show that in many cases, the edge-

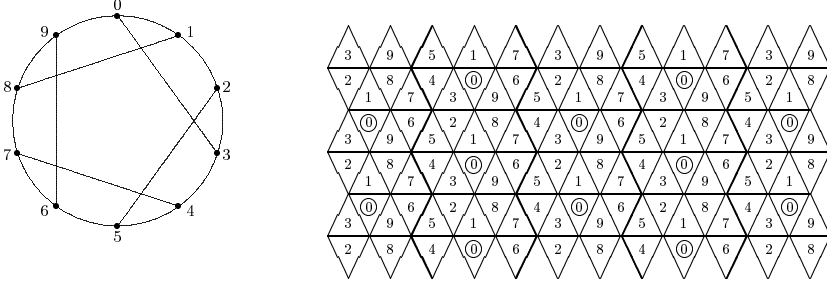


Fig. 1. The graph $C_{10}(1, 9, 3)$ and its associated tessellation. The 0-lattice is the set of triangles containing $\textcircled{0}$.

bisection is the minimum between d and $\frac{2n}{d}$, that is it does not exceed $O(\sqrt{n})$. We have shown that, for any fixed chord, our upper bound on the edge-bisection is optimal up to an $O(\log n)$ factor. Finally, we have given tight bounds for optimal chordal rings, that is those with the maximum number of vertices for a given diameter.

2 Definition of Chordal Rings and Preliminary Results

Chordal rings were first introduced as the graphs obtained from a cycle of even order by adding chords of same odd length matching even and odd vertices (see the left hand side of Figure 1). More formally, we define chordal rings as follows.

Definition 1. Let $n \geq 3$ be any integer, and let d be an odd integer such that $1 < d \leq n$. The chordal ring $C_{2n}(d)$ of order $2n$ and chord d is the graph obtained by the union of the cycle C_{2n} of order $2n$ with the set of edges $\{(2i, 2i + d \bmod 2n), i = 0, 1, \dots, n-1\}$.

The next lemma gives a trivial upper bound on the edge-bisection of $C_{2n}(d)$.

Lemma 1. The edge-bisection of $C_{2n}(d)$ is at most $d + 2$.

Proof. Let us partition the vertex set $\{0, \dots, 2n-1\}$ into the two sets $\{0, \dots, n-1\}$ and $\{n, \dots, 2n-1\}$. One can easily check that the set of edges separating the two parts has cardinality $d + 1$ if n is even, and $d + 2$ if n is odd.

The notations in Lemmas 2 and 3 are based on the following generalization. Let $n \geq 3$ be any integer, and let a , b , and c be three distinct odd integers in $\{1, \dots, 2n-1\}$. The *generalized chordal ring* of order $2n$ and chords a , b , and c , denoted by $C_{2n}(a, b, c)$, is the graph with vertex set $\mathbb{Z}_{2n}$, in which every vertex $2i$ is connected to vertices $2i + a$, $2i + b$, and $2i + c$. An edge of the form $(i, i + x)$, $x \in \{a, b, c\}$ will be called an x -chord. $C_{2n}(d)$ is isomorphic to $C_{2n}(a, b, c)$ where (a, b, c) is any permutation of $(-1, d, 1)$.

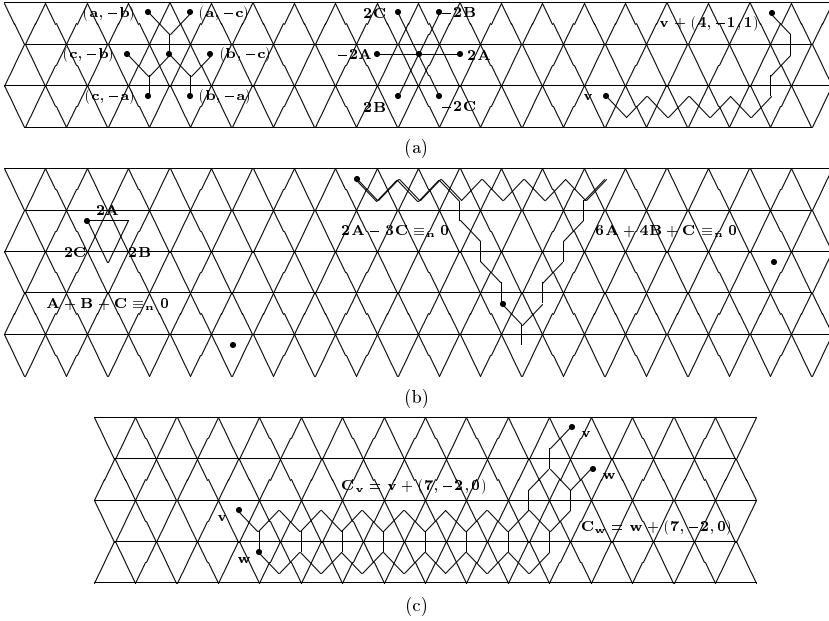


Fig. 2. (a) Representation of paths of even length in a chordal ring. (b) Representation of cycles of $C_{46}(9)$. The 0-lattice associated is the set of triangles containing $\bullet$. (c) Two parallel cycles C_v and C_w in a chordal ring satisfying $7A - 2B \equiv_n 0$.

Generalized chordal rings can be represented by a periodic tessellation of the plane with triangular cells, each cell corresponding to a vertex [20]. The 0-lattice is defined as the set of triangles associated to vertex 0. The example of Figure 1 shows the tessellation and the 0-lattice associated to $C_{10}(1, 9, 3)$. Many fundamental properties of generalized chordal rings can be derived from its plane tessellation. In particular, it allows us to represent paths in the graph as paths in the plane [5].

From any even vertex, there are six different paths of length 2. These paths can be represented by pairs of chords: $(b, -c)$, $(c, -b)$, $(c, -a)$, $(a, -c)$, $(a, -b)$, and $(b, -a)$. A triple of positive integers (x, y, z) will denote the sequence of chords of even length defined by x times the pair of chords $(b, -c)$, then y times the pair of chords $(c, -a)$, and finally z times the pair of chords $(a, -b)$. One can extend this notation to any triple of integers. If x (respectively y or z) is negative, the sequence contains $-x$ times the pair of chords $(c, -b)$ (respectively $-y$ times the pair of chords $(a, -c)$, or $-z$ times the pair of chords $(b, -a)$). For any even vertex v , such a sequence gives a path starting at v that is denoted by $v + (x, y, z)$. Figure 2 (a) shows an example of a path of even length, where $2A = b - c$, $2B = c - a$, and $2C = a - b$.

The path $v + (x, y, z)$ is a cycle if and only if $xA + yB + zC \equiv_n 0$, where $\equiv_n$ denotes the equality modulo n . From the definition of A , B , and C , the

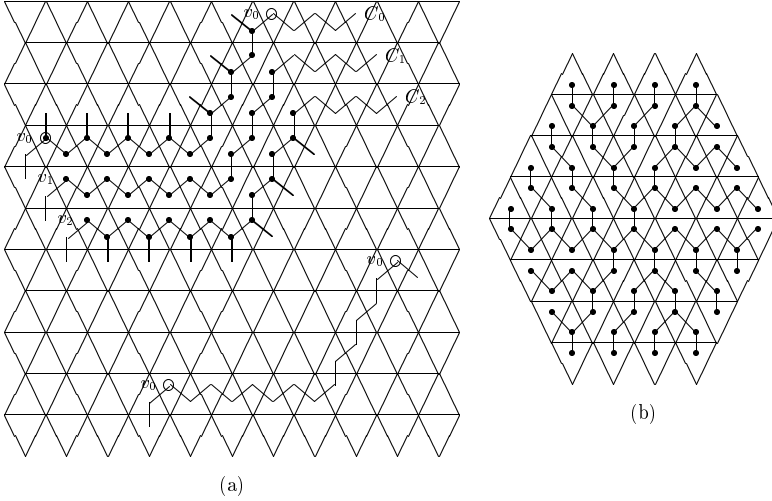


Fig. 3. (a) Construction of a bisector by using parallel cycles. (b) A balanced routing scheme for optimal chordal rings.

equation $A + B + C = 0$ holds. This allows us to simplify any relation of the form $x A + y B + z C \equiv_n 0$ by a relation involving only two of the three integers A , B , and C , whose coefficients are of different signs. See Figure 2 (b) for an example of representation of cycles in the plane.

Two cycles are said *parallel* if they follow the same directions in the tessellation, and if their origins v and w are at distance 2. An illustration of this notion is depicted on (c) of Figure 2.

We have the following:

Lemma 2. *Let v be an even vertex of a generalized chordal ring $C_{2n}(a, b, c)$, and let $w = v - 2C$. Let x and y be two positive integers such that $x A - y B \equiv_n 0$. Then, the parallel cycles $C_v = v + (x, -y, 0)$ and $C_w = w + (x, -y, 0)$ are connected by all a -chords with one end-vertex in*

$$\{w, w + 2A, \dots, w + (x - 1)2A\} \subset C_w,$$

and by all b -chords with one end-vertex in

$$\{v + x 2A, v + x 2A - 2B, \dots, v + x 2A - (y - 1)2B\} \subset C_v.$$

The proof is straightforward, and therefore it is omitted.

The cycles given in the statement of Lemma 2 are in direction $(A, -B)$. If x and y are two positive integers such that $x B - y C \equiv_n 0$, or such that $x A - y C \equiv_n 0$, then similar constructions can be done for the pairs of directions $(B, -C)$ and $(A, -C)$.

Lemma 3. Let $C_{2n}(a, b, c)$ be a generalized chordal ring. Let x and y be two positive integers such that

$$x A - y B \equiv_n 0, \quad \text{or} \quad x B - y C \equiv_n 0, \quad \text{or} \quad x C - y A \equiv_n 0.$$

Then, the edge-bisection width of $C_{2n}(a, b, c)$ is at most $2(x + y) + 1$.

Sketch of the proof. The proof is based on Lemma 2. Assume that $x A - y B \equiv_n 0$ and $n = \alpha 2(x + y) + \beta$ with $0 \leq \beta < 2(x + y)$. Then we can construct a bisector of the graph of size $2(x + y)$ if β is even and $2(x + y) + 1$ if β is odd.

An example of this construction is given on Figure 3 (a). The graph drawn on this figure is $C_{84}(5, 19, 1)$. It satisfies $4 A - 3 B \equiv_n 0$, and it has an edge-bisector of 14 edges.

3 Upper Bounds

Let $C_{2n}(d) = C_{2n}(-1, d, 1)$ be any chordal ring. Therefore, with the notations of the previous sections, we have

$$2A = d - 1, \quad 2B = 2, \quad \text{and} \quad 2C = -(d + 1).$$

In order to apply Lemma 3, we will study the equations $x A - y B \equiv_n 0$, $x B - y C \equiv_n 0$, and $x A - y C \equiv_n 0$, and look for solutions x and y such that $x + y$ is as small as possible. From the values of A , B , and C , we get $A - \frac{d-1}{2} B = 0$. Then, by applying Lemma 3, we obtain the upper bound $d + 2$, already stated in Lemma 1. We can do better as follows. Let us fix the origin of the plane at an arbitrary point of the 0-lattice. Now, let us look for the shortest path from the origin to any point of the 0-lattice. By a symmetry argument we can consider only the pairs of directions $(A, -B)$, $(B, -C)$, and $(A, -C)$.

Let us divide the order $2n$ by $2A = d - 1$. We obtain q and r , the positive integers satisfying $2n = q(d - 1) + r$, $0 \leq r \leq d - 3$, and r even. The division of the order $2n$ by $-2C = d + 1$ gives q' and r' , the positive integers satisfying $2n = q'(d + 1) + r'$, $0 \leq r' \leq d - 1$, and r' even. Actually we have $q A + \frac{r}{2} B = 0$, $\frac{r'}{2} B - q' C = 0$, and $q' \leq q$. Therefore the paths $(q, \frac{r}{2}, 0)$ and $(0, \frac{r'}{2}, -q')$ are both cycles. See Figure 4 for a representation of these cycles. The shortest path from the origin to any point of the 0-lattice depends on the difference $q - q'$.

Since $q = \lfloor \frac{2n}{d-1} \rfloor$ and $q' = \lfloor \frac{2n}{d+1} \rfloor$, the following theorem implies that, for $q \neq q'$, the edge-bisection of $C_{2n}(d)$ is $O(\sqrt{n})$.

Theorem 1. Let k be the edge-bisection width of $C_{2n}(d)$.

1. If $q' = q$, then $k \leq \min\{2q' + r' + 1, d + 2 - r'\}$;
2. If $q' = q - 1$ and $2q < d + 1$ then $k \leq \begin{cases} 2q' + 1 & \text{if } r' = 0; \\ 2q + 1 & \text{otherwise;} \end{cases}$
3. Otherwise, $k \leq d + 2$.

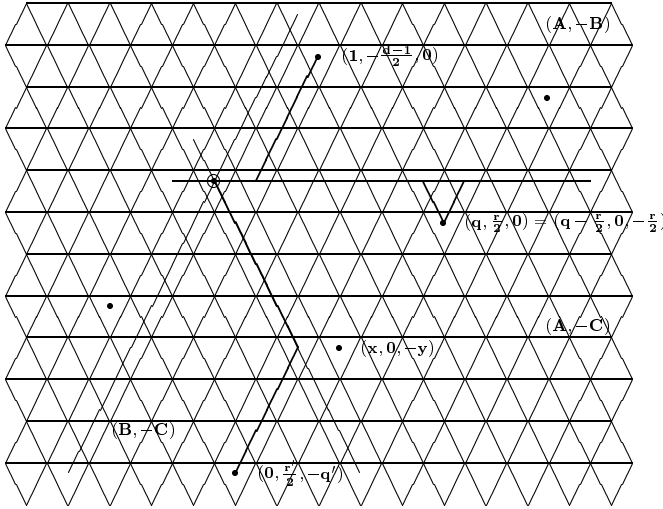


Fig. 4. Representation of the cycles of $C_{38}(7)$. The triangle containing $\otimes$ is the origin, whereas the 0-lattice is the set of triangles containing $\bullet$. The integers x and y in the triple $(x, 0, -y)$ satisfy $x(d-1) + y(d+1) = 2n$ and $x > q$.

Proof. Let us first assume that $q' = q$. Then, from $q(d-1) + r = q(d+1) + r'$, we get $r = 2q' + r' \leq d-1$. Thus the triples $(q, \frac{r}{2}, 0)$ and $(0, \frac{r'}{2}, -q')$ represent the same triangle. In order to apply Lemma 3, it is enough to consider only the two triples $(1, -\frac{d-1}{2}, 0)$, $(0, \frac{r'}{2}, -q')$, and their sum $(1, \frac{-d+1+r'}{2}, -q')$.

One can check that any other triple would give a larger upper bound. Since, in this case $1 > -q' > \frac{-d+1+r'}{2}$, the triple $(1, \frac{-d+1+r'}{2}, -q')$ can be reduced to $(q' + 1, \frac{-d+1+r'+2q'}{2}, 0)$. From Lemma 3, the upper bounds given by these three triples are $d+2$, $2q' + r' + 1$, and $d+2-r'$. Therefore $k \leq \min\{2q' + r' + 1, d+2-r'\}$.

Now, let us assume that $q' = q - 1$. Then, the three triples to consider are $(1, -\frac{d-1}{2}, 0)$, $(q, \frac{r}{2}, 0)$, and $(0, \frac{r'}{2}, -q')$. Since in this case $q(c-1) + r = (q-1)(d+1) + r'$, we have $r = 2q + r'$ which implies $2q \leq r$. So, $(q, \frac{r}{2}, 0)$ can be reduced to $(q - \frac{r}{2}, 0, -\frac{r}{2})$. From Lemma 3, the upper bounds given by these three triples are $d+2$, $2q+1$, and $2q' + r' + 1$. The smallest value of these three upper bounds is $2q' + 1$ if $2q < d+1$ and $r' = 0$; $2q+1$ if $2q < d+1$ and $r' \neq 0$; $d+2$ if $2q \geq d+1$.

Finally, if $q - q' \geq 2$, then the equation $q(d-1) + r = q'(d+1) + r'$ implies $2q > d+1 > r$, and $2q' + r' > d+1$. Thus, the upper bounds given by $(q - \frac{r}{2}, 0, -\frac{r}{2})$ and $(0, \frac{r'}{2}, -q')$ are $2q+1$ and $2q' + r' + 1$, respectively. They are both larger than $d+2$. Other cycles to consider are those represented by the triples $(x, 0, -y)$, with x and y positive integers, and $x(d-1) + y(d+1) = 2n$. One can check that, for these cycles, $x > q$. Therefore, the smallest upper bound on k given by the application of Lemma 3 is $d+2$. See an example of this case on Figure 4.

4 Lower Bounds

In this section we give lower bounds on the edge-bisection of chordal rings. These lower bounds are derived from the comparison between the number of rounds of a gossip protocol (all-to-all broadcasting) on chordal rings described in [6], and a lower bound on the gossip complexity given in [15].

It is stated in [15] that the minimum number of rounds $g(G)$ required for gossiping in a graph G of order N and edge-bisection k satisfies $2\lceil\log_2 N\rceil - \lceil\log_2 k\rceil - \lceil\log_2 \log_2 N\rceil - 2 \leq g(G)$. The number of rounds of the gossip protocol for $C_{2n}(d)$ described in [6] is $2\lceil\log_2 2n\rceil - \lceil\log_2 b\rceil + O(1)$, where $b = \min\{d, q'\}$. Therefore, we conclude that $\lceil\log_2 k\rceil \geq \lceil\log_2 b\rceil - \lceil\log_2 \log_2(2n)\rceil + O(1)$.

Thus, the edge-bisection k of $C_{2n}(d)$ satisfies $k = \Omega(\frac{b}{\log_2 n})$. Let us compare this lower bound with the upper bound of Theorem 1.

Case 1 of Theorem 1 corresponds to $q' = q$. This implies $q' < d$, and thus there exists $\alpha \geq 0$ such that

$$\alpha \frac{q'}{\log_2 n} \leq k \leq \min\{2q' + r' + 1, d + 2 - r'\}.$$

Case 2 of Theorem 1 corresponds to $q' = q - 1$ and $2q < d + 1$. This implies $q' < c$, and thus there exists $\alpha \geq 0$ such that

$$\alpha \frac{q'}{\log_2 n} \leq k \leq 2q' + 3.$$

Case 3 of Theorem 1 corresponds to $q' = q - 1$ and $2q \geq d + 1$, or $q - q' > 1$. One can check that either $d > q' > \frac{d-1}{2}$, or $q' \geq d$. In both cases, there exists $\alpha \geq 0$ such that

$$\alpha \frac{d}{\log_2 n} \leq k \leq d + 2.$$

Therefore, in Case 2 and Case 3, the upper bounds on the edge-bisection given in Theorem 1 are optimal within a factor of $\log_2 n$. Actually, we have proved the following:

Theorem 2. *Given any positive odd integer d , the upper bound on the edge-bisection of $C_{2n}(d)$ given in Theorem 1 is optimal up to an $O(\log n)$ factor.*

Note that we cannot show that the bound of Case 1 of Theorem 1 is optimal up to an $O(\log n)$ factor by using the gossip argument. However, it is worth mentioning that one can apply other techniques to derive tight bounds for a very important subclass of chordal rings: a chordal ring is called *optimal* if it has the maximum number of vertices for a given diameter.

It is known that, for every positive odd integer D , the optimal chordal ring of diameter D is $C_{2n}(3D)$ with $n = \frac{3D^2+1}{4}$ [20]. The graphs in this subclass are exactly the edge-transitive generalized chordal rings [5]. Figure 3 (b) represents a set of triangles containing all the vertices of the graph exactly once. The symmetry of this pattern, induced by the edge-transitivity of the graph, allows us to define a balanced routing, as shown in Figure 3 (b).

From the values $2n = \frac{3D^2+1}{2}$ and $d = 3D$, we have the equalities

$$2n = \frac{D-1}{2}(d-1) + 2D = \frac{D-1}{2}(d+1) + D+1,$$

that is, $q = q' = \frac{D-1}{2}$, $r = 2D$, and $r' = D+1$. Thus optimal chordal rings satisfy the conditions of Case 1 of Theorem 1. We have proved the following:

Theorem 3. *Let D be any positive odd integer, and let k be the edge-bisection of the optimal chordal ring of diameter D . Then*

$$\frac{27}{16}D + O\left(\frac{1}{D}\right) \leq k \leq 2D.$$

Sketch of the proof. To prove that theorem, we used the standard method presented in [17] based on the fact that, if the complete graph can be embedded in a graph G with congestion at most ℓ , then the edge-bisection of G is at least $\frac{n^2}{4\ell}$. In other words, if the edge-forwarding index [10] of G is $\pi(G)$ then the edge-bisection of G is at least $\frac{n^2}{4\pi(G)}$ (recall that the edge-forwarding index of a graph G is defined as the minimum, over all embedding of the complete graph in G , of the congestion of the embedding). By using the routing scheme of Figure 3 (b) we can compute the following upper bound for the edge-forwarding index of $C_{2n}(3D)$:

$$\pi(C_{2n}(3D)) \leq \frac{2}{3}D^3 + \frac{1}{3}D + O(1).$$

This gives, for the edge-bisection, the lower bound $\frac{27}{16}D + O(\frac{1}{D}) \leq k$. The upper bound $k \leq 2D$ follows from Theorem 1, with $q = q' = \frac{D-1}{2}$, $r = 2D$, and $r' = D+1$.

Unfortunately, it is not always easy to compute lower bounds in this way because we have to define good embeddings in order to obtain good upper bounds on $\pi(C_{2n}(d))$. The computation of the edge-forwarding index is difficult in general, even for very symmetric graphs [12,14]. Actually, computing the edge-forwarding index of an arbitrary chordal ring is a challenging problem. Therefore, even though we believe that the bounds given in Theorem 1 are tight, we let this question as an open problem.

Acknowledgements. The authors are thankful to Pierre Fraigniaud for his help to improve the presentation of the results, and to Josep Burillo and Ralf Klasing for their valuable comments.

References

1. A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.
2. F. Annexstein and M. Baumslag. On the diameter and bisector size of Cayley graphs. *Math. Systems Theory*, 26:271–291, 1993.

3. B. Arden and H. Lee. Analysis of chordal ring network. *IEEE Trans. Comput.*, C-30(4):291–295, April 1981.
4. S. Arora, D. Karger, and M. Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC-95)*, pages 284–293. ACM Press, 1995.
5. L. Barrière. Triangulations and chordal rings. In *6th Int. Colloquium on Structural Information and Communication Complexity (SIROCCO)*, volume 5 of *Proceedings in Informatics*, pages 17–31. Carleton Scientific, 1999.
6. L. Barrière, J. Cohen, and M. Mitjana. Gossiping in chordal rings under the line model. In J. Hromkovic and W. Unger, editors, *Proceedings of the MFCS'98 Workshop on Communication*, pages 37–47, 1998.
7. S. Blackburn. Node bisectors of Cayley graphs. *Mathematical Systems Theory*, 29:589–598, 1996.
8. R. Boppana. Eigenvalues and graph bisection: An average-case analysis. In Ashok K. Chandra, editor, *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, pages 280–285, Los Angeles, CA, October 1987. IEEE Computer Society Press.
9. T. Bui, S. Chaudhuri, T. Leighton, and M. Sipser. Graph bisection algorithms with good average case behavior. In *25th Annual Symposium on Foundations of Computer Science*, pages 181–192, Los Angeles, Ca., USA, October 1984. IEEE Computer Society Press.
10. D. Dolev, J. Halpern, B. Simons, and R. Strong. A new look at fault tolerant network routing. In *Proc. ACM 16th STOC*, pages 526–535, 1984.
11. W. Donath and A. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17:420–425, 1973.
12. S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal of Computing*, 5(4):691–703, 1976.
13. N. Garey and D. Johnson. *Computers and Intractability - A Guide to Theory of NP-completeness*. W.H. Freeman and Company, 1979.
14. M.-C. Heydemann, J.-C. Meyer, J. Opatrny, and D. Sotteau. Forwarding indices of consistent routings and their complexity. *Networks*, 24:75–82, 1994.
15. J. Hromkovic, R. Klasing, W. Unger, and H. Wagener. Optimal algorithms for broadcast and gossip in edge-disjoint path modes. In (*SWAT'94*), *Lecture Notes in Computer Science 824*, 1994.
16. B. Kerningham and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Systems Technology Journal*, 49:291–307, 1970.
17. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*. Morgan Kaufmann Publishers, 1992.
18. R. Lipton and R. Tarjan. A separator theorem for planar graphs. In *WATERLOO: Proceedings of a Conference on Theoretical Computer Science*, 1977.
19. Jerrum M and G. Sorkin. Simulated annealing for graph bisection. In *34th Annual Symposium on Foundations of Computer Science*, pages 94–103, Palo Alto, California, 3–5 November 1993. IEEE.
20. P. Morillo. *Grafos y digrafos asociados con teselaciones como modelos para redes de interconexión*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 1987.
21. D. Robinson. *A Course in Theory of Groups*, volume 80 of *Graduate Texts in Math*. Springer, 1996.

Equation Satisfiability and Program Satisfiability for Finite Monoids

David Mix Barrington¹, Pierre McKenzie², Cris Moore³, Pascal Tesson⁴, and Denis Thérien^{4*}

¹ Dept. of Computer Science, University of Massachusetts

² Dept. d'Informatique et de Recherche Opérationnelle, Université de Montréal

³ Dept. of Computer Science, University of New Mexico

⁴ School of Computer Science, McGill University

Abstract. We study the computational complexity of solving equations and of determining the satisfiability of programs over a fixed finite monoid. We partially answer an open problem of [4] by exhibiting quasi-polynomial time algorithms for a subclass of solvable non-nilpotent groups and relate this question to a natural circuit complexity conjecture. In the special case when M is aperiodic, we show that PROGRAM SATISFIABILITY is in P when the monoid belongs to the variety **DA** and is NP-complete otherwise. In contrast, we give an example of an aperiodic outside **DA** for which EQUATION SATISFIABILITY is computable in polynomial time and discuss the relative complexity of the two problems. We also study the closure properties of classes for which these problems belong to P and the extent to which these fail to form algebraic varieties.

1 Introduction

In [4], Goldmann and Russell investigated the computational complexity of determining if an equation over some fixed finite group has a solution. Formally, an equation over a group, or more generally over a monoid M , is given as

$$c_0 X_{i_1} c_1 \dots c_{n-1} X_{i_n} c_n = m$$

where the $c_i \in M$ are constants, $m \in M$ is the target and the X_i 's are variables, not necessarily distinct. The EQUATION SATISFIABILITY problem for M (which we will denote EQN-SAT $_M$) is to determine if there is an assignment to the variables such that the equation is satisfied. It is shown in [4] that this problem is NP-complete for any non-solvable group but lies in P for nilpotent groups. To prove the latter result, they introduced the harder problem of determining whether a given program over a group G outputs some $g \in G$.

* P. McKenzie, P. Tesson and D. Thérien are supported by NSERC and FCAR grants. A part of the work was completed during workshops held respectively by DIMACS-DIMATIA (June 99) and McGill University (February 00). The authors wish to thank the organizers of both events.

We complete and extend their results by considering the computational complexity of both Equation Satisfiability and Program Satisfiability over a fixed monoid M . When M is a group, we show that lower bounds on the length of programs over M that can compute AND yield upper bounds on the time complexity of PROG-SAT and EQN-SAT. In particular, we give a quasi-polynomial time algorithm for a class of solvable, non-nilpotent groups. We also uncover a tight relationship between the complexity of PROG-SAT for groups and the conjecture that any bounded-depth circuit built solely of MOD_m gates requires exponential size to compute AND [2].

When M is an aperiodic monoid, we prove that PROG-SAT is solvable in polynomial time when M belongs to the variety **DA** and is NP-complete otherwise. This is in sharp contrast with the fact that there are aperiodics lying outside **DA** for which EQN-SAT is in P and others for which it is NP-complete.

We also investigate the complexity of #PROG-SAT and show that it is #P-complete for aperiodics outside of **DA** and non-solvable groups, but that it is in #L for monoids in **DA** and not #P-complete under polynomial-time transformations for a subclass of solvable groups, including nilpotent groups.

Finally, we discuss the relative complexity of PROG-SAT and EQN-SAT and the closure properties of classes of monoids for which these problems are in P. Section 2 outlines the necessary background on finite monoids and programs over monoids. Sections 3 and 4 describe our results for groups and aperiodic monoids respectively. Finally, the differences among the two problems are explored in Section 5 and open questions are presented in the conclusion.

2 Preliminaries

2.1 Finite Monoids

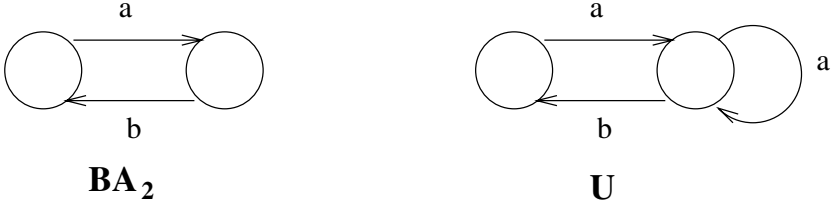
A monoid M is a set with a binary associative operation and an identity. This operation defines a canonical surjective morphism $\text{eval}_M : M^* \rightarrow M$ by $\text{eval}_M(m_1 m_2 \dots m_k) = m_1 \cdot m_2 \cdots m_k$ which just maps a sequence of monoid elements to their product in M . In this paper, we will only be considering finite monoids, with the exception of the free monoid A^* .

We say that a monoid N divides monoid M and write $N \prec M$ if N is the homomorphic image of a submonoid of M . A class of finite monoids $\mathcal{M}$ is said to be a (pseudo-)variety if it is closed under direct product and division. Varieties are the natural way of classifying finite monoids.

A monoid M is aperiodic or group-free if no subset of it forms a non-trivial group or, equivalently, if it satisfies $m^{t+1} = m^t$ for some integer t and all $m \in M$. We will also be interested in the subvariety of aperiodics called **DA**. The monoids in **DA** are the aperiodics satisfying $(stu)^n t (stu)^n = (stu)^n$, for some $n \geq 0$. It is known that M belongs to **DA** iff for all $F \subseteq M$ the set $\{w \in M^* : \text{eval}_M(w) = m \in F\}$ can be expressed as the disjoint union of languages of the form $L = A_0^* a_1 A_1^* a_2 \dots a_k A_k^*$, where $a_i \in M$ and $A_i \subseteq M$, and this concatenation is unambiguous, meaning that if $w \in L$ then there is a unique way of factoring w

as $w_0 a_1 w_1 \dots a_k w_k$ with $w_i \in A_i^*$ [7]. This variety is omnipresent in investigations of complexity-related questions on finite monoids.

We will also be particularly concerned with two aperiodic monoids BA_2 and U that do not belong to **DA**. They are the transition monoids associated with the following partial automata (missing arcs map to an implicit sink state):



Both BA_2 and U have 6 elements (namely $\{1, a, b, ab, ba, 0\}$) but in BA_2 we have $a^2 = b^2 = 0$ while in U we have $b^2 = 0$ but $a^2 = a$. Note also that for both monoids we have $bab = b$ and $aba = a$. It is known that for any finite aperiodic monoid M that does not belong to **DA** either $U \prec M$ or $BA_2 \prec M$. These two monoids are thus minimal examples of aperiodics outside **DA**.

The wreath product of two monoids M, N , denoted $M \circ N$, is the set $M^N \times N$ with an operation defined as $(f_1, n_1) \cdot (f_2, n_2) = (f_1 f', n_1 n_2)$ where $f' : N \rightarrow M$ is defined as $f'(x) = f_2(xn_1)$. Note that the product xn_1 is well defined since $x \in N$. A group G is solvable iff it divides the wreath product of cyclic groups. (A proof of this fact and a detailed introduction to finite monoids and algebraic automata theory can be found e.g. in [3].)

2.2 Programs over Monoids

The formalism of programs over finite monoids (or non-uniform DFA's as they were first called) was introduced by Barrington [1] to show the equivalence of NC^1 and bounded-width branching programs of polynomial length.

An n -input program over M is a sequence of instructions

$$\phi = (i_1, s_1, t_1)(i_2, s_2, t_2) \dots (i_l, s_l, t_l)$$

where the $i_j \in [n]$ are bit positions in the input and $s_j, t_j \in M$. Given an input $x \in \{0, 1\}^n$, the program ϕ outputs the string $\phi(x) = w_1 w_2 \dots w_l$ where $w_j = s_j$ if the i_j^{th} bit of the input is 0 and $w_j = t_j$ if this bit is 1. We say that a language $L \subseteq \{0, 1\}^n$ is recognized by this program if there exists $F \subseteq M$ such that x is in L iff $eval_M(\phi(x))$ is in F .

There is a deep connection between such programs and subclasses of NC^1 . In particular, a language is in NC^1 iff it can be recognized by a polynomial length program over a finite monoid. Similarly AC^0 is exactly the class of languages recognizable by polynomial length programs over aperiodic monoids and CC^0 circuits (polynomial size bounded depth circuits using only MOD_m gates) correspond to programs of polynomial length over solvable groups. A survey of such results can be found in [5].

Given an n -input program ϕ over a monoid M and a target set $F \subseteq M$, the PROGRAM SATISFIABILITY problem (denoted PROG-SAT_M) is to determine whether there exists $x \in \{0, 1\}^n$ such that $\phi(x)$ evaluates to $m \in F$. We will also study the corresponding counting problem $\#\text{PROG-SAT}$.

Determining the satisfiability of a program over M is always at least as hard as determining the satisfiability of equations over the same monoid.

Lemma 1. *For any finite monoid M , $\text{EQN-SAT}_M \leq_P \text{PROG-SAT}_M$.*

Proof. Suppose the equation $c_0 X_{i_1} c_1 \dots X_{i_n} c_n = m$ has t variables. It can be satisfied iff the following M -program over $t \cdot |M|$ variables can be satisfied: we replace every constant c_i by the instruction $(1, c_i, c_i)$ and each occurrence of the variable X_i by a sequence of $|M|$ instructions querying variables $Y_{j_1}, \dots, Y_{j_{|M|}}$ of the form $(Y_{j_1}, 1, m_1)(Y_{j_2}, 1, m_2) \dots (Y_{j_{|M|}}, 1, m_{|M|})$ where $m_1 \dots m_{|M|}$ are the elements of M . Note that for any $m \in M$, there is an assignment of the Y_j 's such that this sequence of $|M|$ instructions evaluates to m .

As we will see, the converse of this fact is not true and we will highlight the differences in the complexity of these two similar problems.

Note that throughout these notes, we (shamelessly) assume that $P \neq NP$.

3 Groups

It was shown by Goldmann and Russell in [4] that EQN-SAT_G is NP-complete for any non-solvable group G . They also proved that both EQN-SAT and PROG-SAT can be solved in polynomial time for a nilpotent group. Using Lemma 1, we thus know that PROG-SAT_G is also NP-complete for non-solvable groups. It is much easier, however, to prove this directly using well known properties of programs over these groups.

Theorem 1. *For any non-solvable group G , we have PROG-SAT_G is NP-complete and $\#\text{PROG-SAT}_G$ is $\#P$ -complete.*

This follows from the fact [1] that given a boolean formula $\psi(X_1, X_2, \dots, X_n)$, we can, for any non-solvable G build an n -input program $\phi(X_1, X_2, \dots, X_n)$ whose length is polynomially related to the length of ψ and such that $\phi(b_1, b_2, \dots, b_n) = 1$ iff the truth assignment $(b_1, b_2, \dots, b_n)$ satisfies ψ . We can thus transform SAT and $\#\text{SAT}$ to PROG-SAT_G and $\#\text{PROG-SAT}_G$ respectively.

The complexity of both EQN-SAT and PROG-SAT for solvable non-nilpotent groups is closely tied to whether or not there exist programs over these groups computing the AND function in sub-exponential length. We will say that a finite group G is AND-strong if there exists a family of polynomial length programs over G computing AND. We will say that G is AND-weak if programs over G computing the AND of t variables require length $2^{\Omega(t)}$.

As we have mentioned, all non-solvable groups are provably AND-strong, but the only groups known to be AND-weak are groups which divide the wreath

product of a p -group by an abelian group. In particular, these include S_3 and A_4 . It has been conjectured, however, that all solvable groups are AND-weak or, equivalently, that CC^0 circuits require exponential size to compute AND (see [2] for a detailed discussion).

Theorem 2. *If G is AND-weak then $PROG-SAT_G$ is solvable in quasi-polynomial time.*

Proof. We claim that if a program in s variables over G can be satisfied, then it can be satisfied by an assignment of weight logarithmic in the length of the program. Suppose not. We define the weight $|x|_1$ of $x \in \{0, 1\}^*$ as the number of 1's in x . Let w be a satisfying assignment of minimal weight, with $|w|_1 = k$. Assume W.L.O.G. that the first k bits of w are 1. By fixing $X_{k+1}, \dots, X_s$ to 0, we obtain a program over G which is computing the AND of k bits because w was assumed to have minimal weight. Since G is AND-weak, we must have $n \geq 2^{\Omega(k)}$, so $k \leq O(\log n)$. It is thus sufficient to consider only the $O(\binom{n}{O(\log n)}) = O(n^{O(\log n)})$ assignments of weight at most k , so we have a quasi-polynomial time algorithm.

Similarly, $PROG-SAT_G$ can be shown [4] to be in P for a nilpotent group G by using the fact [2,6] that a satisfiable program over a nilpotent group can be satisfied by an assignment of weight bounded by a constant independent of the program's length.

Since we do not know that all solvable groups are AND-weak, we would like to give results on the complexity of EQN-SAT and PROG-SAT for solvable groups which are unrelated to their computational power. However, the following theorem shows how closely tied these two questions really are.

Theorem 3. *If G is AND-strong, then $PROG-SAT$ is NP-complete for the wreath product $G \circ Z_k$ for any cyclic group Z_k of order $k \geq 4$.*

Proof. We want to build a reduction from 3-SAT. Define the function $f_{g_0, g_1} : Z_k \rightarrow G$ as

$$f_{g_0, g_1}(x) = \begin{cases} g_0 & \text{if } x = 0 \\ g_1 & \text{if } x \neq 0 \end{cases}$$

Also, denote by id the function such that $id(x) = 1_G$ for all $x \in Z_k$. Consider now the following 3-input program over $G \circ Z_k$

$$\begin{aligned} \phi = & (1, (id, 0), (id, 1)) (2, (id, 0), (id, 1)) (3, (id, 0), (id, 1)) (1, (\mathbf{f}_{g_0, g_1}, \mathbf{0}), (\mathbf{f}_{g_0, g_1}, \mathbf{0})) \\ & (1, (id, 0), (id, -1)) (2, (id, 0), (id, -1)) (3, (id, 0), (id, -1)) \end{aligned}$$

First note that the Z_k component of ϕ 's output will always be 0. Note also that the middle instruction's output is independent of the value of the bit queried. It is also the only instruction affecting the G^{Z_k} component of the output. This component is a function f such that $f(0) = g_1$ if one of the input bits is on and $f(0) = g_0$ otherwise. To see this note that when we execute the middle

instruction, the product in Z_k so far is not equal to zero iff one of the instructions yielded a +1. Thus, ϕ is recognizing the OR of these three variables.

Suppose the 3-SAT instance has m clauses. By assumption there is a G -program of length m^c that computes the AND of m variables. If we replace every instruction (i, g_0, g_1) by a program over $G \circ Z_k$ as above, we obtain a program of length $7 \cdot m^c$ which is satisfiable iff the 3-SAT instance is satisfiable.

The wreath product of a solvable group by an abelian group is also solvable and, as we can see from the proof of Theorem 2, super-polynomial lower bounds on the length of programs recognizing the AND over a group G translate into sub-exponential upper bounds on the time complexity of PROG-SAT_G . Thus, assuming that no sub-exponential time algorithm can solve an NP-hard problem, there exists an AND-strong solvable group iff there exists a solvable group for which PROG-SAT is NP-complete.

We have seen that $\#\text{PROG-SAT}$ is $\#\text{P}$ -complete for non-solvable groups. In contrast, that problem is not $\#\text{P}$ -complete under parsimonious reductions for any nilpotent group. Indeed, a program over a nilpotent group is either unsatisfiable or is satisfied by a constant fraction of the inputs [6]. This immediately disqualifies it as a $\#\text{P}$ -complete problem since most problems in $\#\text{P}$ fail to have this property. Similarly, a polynomial length program over an AND-weak group can not, by definition, have only a single satisfying assignment. $\#\text{PROG-SAT}$ is thus not $\#\text{P}$ -complete under parsimonious reductions for these groups either.

For an abelian group A , counting the number of satisfying assignments to a program is in $\#\text{L}$. The $\#\text{L}$ machine only has to guess the assignment one bit at a time. Then, scanning through the program ϕ it can calculate the effect of this choice on the output of ϕ using commutativity. Similar ideas do not seem to yield good algorithms for $\#\text{PROG-SAT}$ for non-abelian groups.

4 Aperiodic Monoids

In the case of groups, we have no evidence of any significant difference between the complexity of solving equations versus that of satisfying programs. The situation is remarkably different in the case of aperiodic monoids. In this section, we characterize the subclass of aperiodics for which PROG-SAT can be solved in polynomial time but show that there are aperiodics for which PROG-SAT is NP-complete but EQN-SAT belongs to P. We are also able to make explicit the link between the algebraic properties of an aperiodic and the complexity of the corresponding $\#\text{PROG-SAT}$ problem.

Theorem 4. *For any monoid M in the variety $\mathbf{DA}$, PROG-SAT_M is in P.*

Proof. As we mentioned, for any monoid M in $\mathbf{DA}$, the sets $\{w : \text{eval}_M(w) = m \in F \text{ for some } F \subseteq M\}$ can always be expressed as the finite disjoint union of languages of the form $A_0^* a_1 A_1^* \dots a_k A_k^*$, where $a_i \in M$, $A_i \subseteq M$, and the concatenation is unambiguous.

Hence it is sufficient to consider the at most $\binom{l}{k}$ k -tuples of instructions of the program of length l that could be held responsible for the presence of the subword $a_1 a_2 \dots a_k$. For each of them, we need to check if there is an assignment such that the output of the program belongs to $A_0^* a_1 A_1^* \dots a_k A_k^*$ and that can clearly be done in linear time.

Theorem 5. *For any monoid M in the variety $\mathbf{DA}$, $\#PROG-SAT_M$ is in $\#L$.*

Proof. The idea is similar. The $\#L$ machine will guess which k -tuple of instructions yields the bookmarks $a_1, \dots, a_k$. It then non-deterministically guesses the value of each of the n variables and checks that this never makes an instruction sitting between a_i and a_{i+1} output an element outside A_i . Only $(k+1)$ indices are kept in memory throughout the computation. Note that we will not count anything twice since the concatenation $A_0^* a_1 A_1^* \dots a_k A_k^*$ is unambiguous.

Let us look now at the two minimal examples of aperiodic monoids that do not belong to $\mathbf{DA}$.

Theorem 6. *$EQN-SAT$ for BA_2 is NP-complete.*

Proof. We use a reduction from EXACTLY-ONE-IN-3 SAT. Each variable v_i in the 3-SAT instance is represented by two variables v_i^+ and v_i^- representing v_i and its complement in the equation. We build the following equation with target ab . First, we concatenate, for each i the segments $abv_i^+v_i^-bv_i^-v_i^+b$ and for each clause e.g. $(v_i, \bar{v}_j, v_k)$ we concatenate $abv_i^+v_j^-v_k^+b$.

It is easy to see that the first half of the equation forces us to choose one of v_i^+, v_i^- as 1 and the other as a . If we now interpret a as TRUE and 1 as FALSE, the equation is satisfiable iff we can choose assignments to the v_i such that in every segment e.g. $abv_i^+v_j^-v_k^+b$ exactly one of the variables is set to a .

A similar proof will provide the same result for a larger class of aperiodics outside $\mathbf{DA}$. At this point, one might even be tempted to conjecture that $EQN-SAT$ is NP-complete for any aperiodic outside of $\mathbf{DA}$, but this is not the case:

Theorem 7. *$EQN-SAT$ for U can be decided in polynomial time.*

Proof. To show this we will use the fact that, in U , $axa = a$ whenever $x \neq 0$. Intuitively, we use the fact that a 's are our friends. In particular, we have that if $xyz = a$ then $xaz = a$.

We will show that for any target, if the equation is satisfiable then it can be satisfied by an assignment with a very precise structure. We are given the expression $E : c_0 X_{i_1} c_1 \dots X_{i_n} c_n$ and a target m .

If $m = 0$, the equation is trivially satisfiable by setting any variable to 0, and if $m = 1$, it is satisfiable iff all the c_i 's are 1. Since the equation is 0 if any of the c_i is 0, we will assume that the constants are non-zero.

If $m = a$, then E is satisfiable iff it is satisfied when all the variables are set to a , namely when we have both $c_0 \in \{1, a, ab\}$ and $c_n \in \{1, a, ba\}$.

If $m = ba$, and E can be satisfied, then it can be satisfied by one of the $O(n)$ assignments of the following form: all the variables occurring before some point j in the equation (which might be a constant or a variable) are set to 1, the variable at point j is set to ba and the other variables are set to a . To see this, consider any satisfying assignment to E . If the first b in the induced string comes from a constant, then all the variables occurring before it must have been set to 1. Moreover, all we have to insure now is that there are no consecutive b 's in the suffix. So we can set all the variables that haven't yet occurred to a without affecting the target. If the first b occurs in a variable, the same reasoning shows that we can set this variable to ba and the variables not yet considered to a . So it is sufficient to check a linear number of possible assignments to decide satisfiability. The case $m = ab$ is handled in a similar, symmetrical way.

Finally if $m = b$, it suffices to consider the following $O(n^2)$ assignments: variables occurring before some j or after some k are set to the identity, the variable at point j is set to ba or b , the one at point k to ab or b , and all remaining variables are set to a . Again, if we now consider any satisfying assignment and call j and k the first and last occurrence of b in the induced word over M^* , then we know that all variables occurring before j or after k were set to 1. The variable or constant at point j must be b or ba , the one at point k being b or ab so we still have a satisfying assignment if we set the rest of the variables to a .

The example of U also proves that the complexity of EQN-SAT and PROG-SAT can be radically different since we also have:

Theorem 8. *PROG-SAT_U is NP-complete and #PROG-SAT_U is #P-complete.*

Proof. The program $\phi_i = (X_{i_1}, b, ba)(X_{i_2}, 1, a)(X_{i_3}, 1, a) \dots (X_{i_k}, ba, a)$ (over U) outputs ba if one of the X_{i_j} 's is set to 1 and 0 otherwise. By concatenating such ϕ_i 's we get a program whose output is ba if all ϕ_i 's have one of their input variables set to 1 and 0 otherwise. So we can simulate a CNF formula and obtain a reduction from SAT. It is clear that this reduction is parsimonious, hence the #P-completeness result.

From Theorem 6, we know that PROG-SAT_{BA₂} is also NP-complete. It is clear that one can get a parsimonious reduction from EXACTLY-ONE-IN-3 SAT to PROG-SAT_{BA₂} using an idea similar to that of the previous proof. #PROG-SAT_{BA₂} is thus also #P-complete.

As we shall see in the next section, the NP and #P-completeness results for U and BA_2 yield completeness results for all aperiodic monoids outside **DA**.

5 Closure Properties and EQN-SAT vs PROG-SAT

One would expect that when EQN-SAT (or PROG-SAT) is easy to solve for monoid M, N , the problem is also easy for submonoids and homomorphic images of M or N as well as for $M \times N$. However, the previous section provides us with a counterexample to this intuition: EQN-SAT _{$U \times U$} is in P but BA_2 is a submonoid of $U \times U$ with EQN-SAT_{BA₂} NP-complete.

In a perfect world, the classes of monoids for which EQN-SAT and PROG-SAT belong to P would form varieties. Although this is not the case, at least for equations, these classes do have interesting closure properties which further highlight the slight difference in the nature of the two problems.

Theorem 9. *The class $\mathcal{M}_E = \{M : \text{EQN-SAT}_M \in P\}$ is closed under factors (homomorphic images) and direct products.*

In contrast, the class $\mathcal{M}_P = \{M : \text{PROG-SAT}_M \in P\}$ is closed under factors (homomorphic images) and formation of submonoids.

Proof. $\mathcal{M}_E$ is closed under finite direct products since an equation over the product $M \times N$ can simply be regarded as a pair of independent equations for which satisfiability can be checked individually.

For closure under factors, suppose we are given an equation E over the factor N of monoid M . We can build an equation E' over M by replacing each constant by an arbitrary pre-image. If for *any* pre-image of the target we can find an assignment over M satisfying E' then we can map this to a satisfying assignment over N for E . Moreover, if E is satisfiable, then E' must be satisfiable for some pre-image of our original target and there are finitely many pre-images to try.

$\mathcal{M}_P$ is also closed under factors using a similar argument. We simply have to choose representatives for the pre-images of any element of the smaller monoid. Then the program over the larger monoid can be satisfied for at least one pre-image of the original target iff the original program was satisfiable.

Finally, $\mathcal{M}_P$ is closed under submonoids in a trivial way. Note that instructions of an instance of PROG-SAT over the submonoid produce, by definition, elements of the submonoid. The target of this instance is also in the submonoid. So if we can decide satisfiability over the larger monoid, we can naturally use this to decide satisfiability over the submonoid as well.

This theorem justifies our claim of Section 4 that PROG-SAT is NP-complete for all aperiodics outside of **DA**. Indeed, PROG-SAT is NP-complete for both BA_2 and U and any aperiodic outside **DA** has one of them as a divisor. Since the intersection of $\mathcal{M}_P$ with aperiodic monoids is thus **DA**, a variety, it is closed under direct products. However, we do not know how to prove this fact directly.

Intuitively, $\mathcal{M}_E$ is not closed under submonoids because we have no mechanism to guarantee that the variables are only assigned values in the submonoid. The proofs in [4] used the notion of “inducible” subgroups, i.e. subgroups for which such a mechanism does exist. Our results seem to show that this is a necessary evil.

On the other hand, PROG-SAT is easy for submonoids but could be hard for direct products. We can certainly view a program over $M \times N$ as a pair of programs on M and N respectively which are both satisfiable if the original program is, but, conversely, there is no obvious way to check whether the sets of satisfying assignments for each of them are disjoint or not. Thus, polynomial time algorithms for PROG-SAT_M and PROG-SAT_N do not seem to help us get an algorithm for $\text{PROG-SAT}_{M \times N}$, however we have the following partial result:

Theorem 10. *Suppose N is such that $PROG-SAT_N$ is in P , then for any $M \in \mathbf{DA}$, $PROG-SAT_{M \times N}$ is also in P .*

Proof. Let us first look only at the M -component of the program. As in the proof of Theorem 4, we can consider each of the polynomially many k -tuples of instructions that can give rise to the bookmarks $a_1, a_2, \dots, a_k$. For any variable X_i , we can now check if setting it to 0 or 1 causes some instructions to throw us out of the unambiguous concatenation $L = A_0^* a_1 A_1^* \dots a_k A_k^*$. This will force an assignment on some of the variables (or even prove that the target is unreachable given this k -tuple) and leave others free. Given these restrictions, we now look at the program over N obtained after applying these restrictions. By assumption, satisfiability for this program can be checked in P .

The respective closure properties of $\mathcal{M}_E$ and $\mathcal{M}_P$ indicate that beyond their apparent similarities, EQN-SAT and PROG-SAT present different computational challenges.

6 Conclusion

We have shown how the algebraic properties of finite monoids can influence the complexity of solving equations and satisfying programs over them. A few interesting questions still have to be answered. The first is the complexity of PROG-SAT for groups that are solvable but not nilpotent. Although fully resolving this would seem to require new circuit complexity results, we might still be able to use the lower bounds for AND-weak groups to get polynomial-time algorithms. We conjecture that this is not possible, but it would be interesting to find convincing evidence to support this. It is possible that finding good upper bounds for $\#PROG-SAT$ for solvable and nilpotent groups could help in that respect. Because we have to count the number of solutions, we would probably need to move away from the brute force approach used in our algorithms.

We would like to thank Mikael Goldmann and Alex Russell for helpful discussions.

References

1. D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1 . *J. Comput. Syst. Sci.*, 38(1):150–164, Feb. 1989.
2. D. A. M. Barrington, H. Straubing, and D. Thérien. Non-uniform automata over groups. *Information and Computation*, 89(2):109–132, Dec. 1990.
3. S. Eilenberg. *Automata, Languages and Machines*. Academic Press, 1976.
4. M. Goldmann and A. Russell. The complexity of solving equations over finite groups. In *Proceedings of the 14th Annual IEEE Conference on Computational Complexity (CCC-99)*, pages 80–86, 1999.
5. P. McKenzie, P. Péladéau, and D. Thérien. NC^1 : The automata theoretic viewpoint. *Computational Complexity*, 1:330–359, 1991.
6. P. Péladéau and D. Thérien. Sur les langages reconnus par des groupes nilpotents. *Compte-rendus de l'Académie des Sciences de Paris*, pages 93–95, 1988.
7. J.-É. Pin, H. Straubing, and D. Thérien. Locally trivial categories and unambiguous concatenation. *J. Pure Applied Algebra*, 52:297–311, 1988.

XML Grammars

Jean Berstel¹ and Luc Boasson²

¹ Institut Gaspard Monge (IGM), Université Marne-la-Vallée,
F-77454 Marne-la-Vallée Cédex 2
<http://www-igm.univ-mlv.fr/~berstel>

² Laboratoire d'informatique algorithmique: fondements et applications (LIAFA),
Université Denis-Diderot,
F-75251 Paris Cédex 05
<http://www.liafa.jussieu.fr/~lub>

Abstract. XML documents are described by a document type definition (DTD). An XML-grammar is a formal grammar that captures the syntactic features of a DTD. We investigate properties of this family of grammars. We show that an XML-language basically has a unique XML-grammar. We give two characterizations of languages generated by XML-grammars, one is set-theoretic, the other is by a kind of saturation property. We investigate decidability problems and prove that some properties that are undecidable for general context-free languages become decidable for XML-languages.

1 Introduction

XML (eXtensible Markup Language) is a format recommended by W3C in order to structure a document. The syntactic part of the language describes the relative position of pairs of corresponding tags. This description is by means of a document type definition (DTD). In addition to its syntactic part, each tag may also have attributes. If the attributes in the tags are ignored, a DTD appears to be a special kind of context-free grammar. The aim of this paper is to study this family of grammars.

One of the consequences will be a better appraisal of the structure of XML documents. It will also show some limitations of the power of expression of XML. Consider for instance an XML-document that consists of a sequence of paragraphs. A first group of paragraphs is being typeset in bold, a second one in italic, and there should be as many paragraphs in bold than in italic. As we shall see (Example 4.6), it is not possible to specify this condition by a DTD. This is due to the fact that the context-free grammars corresponding to DTD's are rather restricted.

The main results in this paper are two characterizations of XML-languages. The first (Theorem 4.1) is set-theoretic. It shows that an XML-language is the biggest language in some class of languages. It relies on the fact that, for each XML-language, there is only one XML-grammar that generates it. The second characterization (Theorem 4.3) is syntactic. It shows that XML-languages have a kind of "saturation property".

As usual, these results can be used to show that some languages cannot be XML. This means in practice that, in order to achieve some features of pages, additional nonsyntactic techniques have to be used.

The paper is organized as follows. The next section contains the definition of XML-grammars and their relation to DTD. Section 3 contains some elementary results, and in particular the proof that there is a unique XML-grammar for each XML-language. It appears that a new concept plays an important role in XML-languages. This is the notion of surface. The surface of an opening tag a is the set of sequences of opening tags that are children of a . The surfaces of an XML-language must be regular sets, and in fact describe the XML-grammar. The characterization results are given in Section 4. They heavily rely on surfaces, but the second one also uses the syntactic concept of a context.

Section 5 investigates decision problems. It is shown that it is decidable whether the language generated by a context-free language is well-formed, but it is undecidable whether there is an XML-grammar for it. On the contrary, it is decidable whether the surfaces of a context-free grammar are finite. The final section is a historical note. Indeed, several species of context-free grammars investigated in the sixties, such as parenthesis grammars or bracketed grammars are strongly related to XML-grammars. This relationship is sketched.

2 Notation

An XML document [6] is composed of text and of tags. The tags are opening or closing. Each opening tag has a unique associated closing tag, and conversely. There are also tags called empty tags, and which are both opening and closing. In a canonical XML [7], these tags are always replaced by a sequence composed of an opening tag immediately followed by its closing tag. We do so here, and therefore assume that there are no empty tags.

Let A be a set of opening tags, and let $\bar{A}$ be the set of corresponding closing tags. Since we are interested in syntactic structure, we ignore any text. Thus, an XML document is a word over the alphabet $A \cup \bar{A}$.

A document x is *well-formed* if the word x is a correctly parenthesized word, that is if x is in the set of Dyck primes over $A \cup \bar{A}$. Observe that the word is a prime, so it is not a product of two well parenthesized words. Also, it is not the empty word.

An XML-grammar is composed of a terminal alphabet $A \cup \bar{A}$, of a set of variables V in one-to-one correspondence with A , of a distinguished variable called the *axiom* and, for each letter $a \in A$ of a regular set $R_a \subset V^*$ defining the (possibly infinite) set of productions

$$X_a \rightarrow am\bar{a}, \quad m \in R_a, \quad a \in A$$

We also write for short

$$X_a \rightarrow aR_a\bar{a}$$

as is done in DTD's. An XML-language is a language generated by some XML-grammar.

It is well-known from formal language theory that if non-terminals in a context-free grammar are allowed to have infinite regular or context-free sets of productions, then the language generated is still context-free. Thus, any XML-language is context-free.

Example 2.1. The language $\{a^n \bar{a}^n \mid n > 0\}$ is a XML-language, generated by

$$X \rightarrow a(X|\varepsilon)\bar{a}$$

Example 2.2. The language of *Dyck primes* over $\{a, \bar{a}\}$ is a XML-language, generated by

$$X \rightarrow aX^*\bar{a}$$

Example 2.3. The language D_A of *Dyck primes* over $A \cup \bar{A}$ is generated by the grammar

$$\begin{aligned} X &\rightarrow \sum_{a \in A} X_a \\ X_a &\rightarrow aX^*\bar{a}, \quad a \in A \end{aligned}$$

This grammar is not XML. Each X_a in this grammar generates an XML-language, which is $D_A \cap a(A \cup \bar{A})^*\bar{a}$. However D_A is not an XML-language if A has more than one letter.

In the sequel, all grammars are assumed to be *reduced*, that is, every non-terminal is accessible from the axiom, and every non-terminal produces at least one terminal word. Note that, given a grammar with an infinite set of productions, the classical reduction procedure can be applied to get an equivalent reduced grammar when the sets of productions are recursive.

Given a grammar G over a terminal alphabet T and a nonterminal X we denote by

$$L_G(X) = \{w \in (A \cup \bar{A})^* \mid X \xrightarrow{*} w\}$$

the language generated by X in the grammar G .

Remark 2.4. The definition has the following correspondence to the terminology and notation used in the XML community ([6]). The grammar of a language is called a *document type definition* (DTD). The axiom of the grammar is qualified **DOCTYPE**, and the set of productions associated to a tag is an **ELEMENT**. The syntax of an element implies by construction the one-to-one correspondence between pairs of tags and non-terminals of the grammar. Indeed, an element is composed of a *type* and of a *content model*. The type is merely the tag name and the content model is a regular expression for the set of right-hand sides of the productions for this tag. For instance, the grammar

$$\begin{aligned} S &\rightarrow a(S|T)(S|T)\bar{a} \\ T &\rightarrow bT^*\bar{b} \end{aligned}$$

with axiom S corresponds to

```

<!DOCTYPE a [
  <!ELEMENT a ((a|b),(a|b)) >
  <!ELEMENT b (b)* >
]>

```

Here, S and T stand for the nonterminals X_a and X_b respectively.

The regular expressions allowed for the content model are of two types : those called children, and those called mixed. In fact, since we do not consider text, the mixed expressions are no more special expressions.

In the definition of XML-grammars, we ignore *entities*, both general and parameter entities. Indeed, these may be considered as shorthands and are handled at a lexical level.

3 Elementary Results

We denote by D_a the language of *Dyck primes* starting with the letter a . This is the language generated by X_a in Example 2.3. We set $D_A = \cup_{a \in A} D_a$. This is not an XML-language if A has more than one letter. We call D_A the set of Dyck primes over A and we omit the index A if possible. The set D is known to be a *bifix* code, that is no word in D is a proper prefix or a proper suffix of another word in D .

Let L be any subset of the set D of Dyck primes over A . The aim of this section is to give a necessary and sufficient condition for L to be an XML-language.

We denote by $F(L)$ the set of factors of L , and we set $F_a(L) = D_a \cap F(L)$ for each letter $a \in A$. Thus $F_a(L)$ is the set of those factors of words in L that are also Dyck primes starting with the letter a .

Example 3.1. For the language

$$L = \{ab^{2n}\bar{b}^{2n}\bar{a} \mid n \geq 1\}$$

one has $F_a(L) = L$ and $F_b(L) = \{b^n\bar{b}^n \mid n \geq 1\}$.

Example 3.2. Consider the language

$$L = \{a(b\bar{b})^n(c\bar{c})^n\bar{a} \mid n \geq 1\}$$

Then $F_a(L) = L$, $F_b(L) = \{b\bar{b}\}$, $F_c(L) = \{c\bar{c}\}$.

The sets $F_a(L)$ are important for XML-languages and grammars, as illustrated by the following lemma:

Lemma 3.3. *Let G be an XML-grammar over $A \cup \bar{A}$ generating a language L , with nonterminals X_a , for $a \in A$. For each $a \in A$, the language generated by X_a is the set of factors of words in L that are Dyck primes starting with the letter a , that is*

$$L_G(X_a) = F_a(L)$$

Proof. Set $T = A \cup \bar{A}$. Consider first a word $w \in L_G(X_a)$. Clearly, w is in D_a . Moreover, since the grammar is reduced, there are words g, d in T^* such that $X \xrightarrow{*} gX_ad$, where X is the axiom of G . Thus w is a factor of L .

Conversely, consider a word $w \in F_a(L)$ for some letter a , let g, d be a words such that $gwd \in L$. Due to the special form of an XML-grammar, any letter a can only be generated by a production with non-terminal X_a . Thus, a left derivation $X \xrightarrow{*} gwd$ factorizes into

$$X \xrightarrow{k} gX_a\beta \xrightarrow{*} gwd$$

for some word β , where k is the number of letters in g that are in A . Next

$$gX_a\beta \xrightarrow{*} gw'\beta \xrightarrow{*} gwd$$

with $X_a \xrightarrow{*} w'$ and $w' \in D$. None of w and w' can be a proper prefix of the other, because D is bifix. Thus $w' = w$. This shows that w is in $L_G(X_a)$ and proves that $F_a = L_G(X_a)$. $\square$

Corollary 3.4. *For any XML-language $L \subset D_a$, one has $F_a(L) = L$.* $\square$

Let w be a Dyck prime in D_a . It has a unique factorization

$$w = au_{a_1}u_{a_2}\cdots u_{a_n}\bar{a}$$

with $u_{a_i} \in D_{a_i}$ for $i = 1, \dots, n$. The *trace* of the word w is defined to be the word $a_1a_2\cdots a_n \in A^*$.

If L is any subset of D , and $w \in L$, then the words u_{a_i} are in $F_{a_i}(L)$. The *surface* of $a \in A$ in L is the set $S_a(L)$ of all traces of words in $F_a(L)$. Observe that the notion of surface makes sense only for subsets of D .

Example 3.5. For the language of Example 3.1, the surfaces are easily seen to be $S_a = \{b\}$ and $S_b = \{b, \varepsilon\}$.

Example 3.6. The surfaces of the language of Example 3.2 are $S_a = \{b^n c^n \mid n \geq 1\}$ and $S_b = S_c = \{\varepsilon\}$.

It is easily seen that the surfaces of the set of Dyck primes over A are all equal to A^* .

Surfaces are useful for defining XML-grammars. Let $\mathcal{S} = \{S_a \mid a \in A\}$ be a family of regular languages over A . We define an XML-grammar G associated to $\mathcal{S}$ called the *standard grammar* of $\mathcal{S}$ as follows. The set of variables is $V = \{X_a \mid a \in A\}$. For each letter a , we set

$$R_a = \{X_{a_1}X_{a_2}\cdots X_{a_n} \mid a_1a_2\cdots a_n \in S_a\}$$

and we define the productions to be

$$X_a \rightarrow am\bar{a}, \quad m \in R_a$$

for all $a \in A$. Since S_a is regular, the sets R_a are regular over the alphabet V . By construction, the surface of the language generated by a variable X_a is S_a , that is $S_a(L_G(X_a)) = S_a$. For any choice of the axiom, the grammar is an XML-grammar.

Example 3.7. The standard grammar for the surfaces of Example 3.1 is

$$\begin{aligned} X_a &\rightarrow aX_b\bar{a} \\ X_b &\rightarrow b(X_b|\varepsilon)\bar{b} \end{aligned}$$

The language generated by X_a is $\{ab^n\bar{b}^n\bar{a} \mid n \geq 1\}$ and is *not* the language of Example 3.1.

This construction is in some sense the only way to build XML-grammars, as shown by the following proposition.

Proposition 3.8. *For each XML-language L , there exists exactly one reduced XML-grammar generating L , up to renaming of the variables.*

Proof. (Sketch) Let G be an XML-grammar generating L , with nonterminals $V = \{X_a \mid a \in A\}$, and $R_a = \{m \in V^* \mid X_a \rightarrow am\bar{a}\}$ for each $a \in A$. We claim that the mapping

$$X_{a_1}X_{a_2} \cdots X_{a_n} \mapsto a_1a_2 \cdots a_n$$

is a bijection from R_a onto the surface $S_a(L)$ for each $a \in A$. Since the surface depends only on the language, this suffices to prove the proposition. $\square$

It follows easily that

Corollary 3.9. *Let L_1 and L_2 be XML-languages. Then $L_1 \subset L_2$ iff $S_a(L_1) \subset S_a(L_2)$ for all a in A .*

This in turn implies

Proposition 3.10. *The inclusion and the equality of XML-languages are decidable.*

In particular, it is decidable whether an XML-language L is empty. Similarly, it is decidable whether $L = D_\alpha$.

XML-languages are not closed under union and difference. This will be an easy consequence of the characterizations given in the next section (Example 4.8).

Proposition 3.11. *The intersection of two XML-languages is an XML-language.*

4 Two Characterizations of XML-Languages

In this section, we give two characterizations of XML-language. The first (Theorem 4.1) is based on surfaces. It states that, for a given set of regular surfaces, there is only one XML-language with these surfaces, and that it is the maximal language in this family. The second characterization (Theorem 4.3) is syntactical and based on the notion of context.

Let $\mathcal{S} = \{S_a \mid a \in A\}$, be a family of regular languages, and fix a letter a_0 in A . Define $\mathcal{L}(\mathcal{S})$ to be the family of languages $L \subset D_{a_0}$ such that $S_a(L) = S_a$ for all a in A . Clearly, any union of sets in $\mathcal{L}(\mathcal{S})$ is still in $\mathcal{L}(\mathcal{S})$, so there is a maximal language (for set inclusion) in the family $\mathcal{L}(\mathcal{S})$. The *standard* language associated to $\mathcal{S}$ is the language generated by X_{a_0} in the standard grammar of $\mathcal{S}$.

Theorem 4.1. *The standard language associated to $\mathcal{S}$ is the maximal element of the family $\mathcal{L}(\mathcal{S})$. This language is XML, and it is the only XML-language in the family $\mathcal{L}(\mathcal{S})$.*

Example 4.2. The standard language associated to the sets $S_a = \{b\}$ and $S_b = \{b, \varepsilon\}$ of Example 3.1 is the language $\{ab^n b^n \bar{a} \mid n \geq 1\}$ of Example 3.7. Thus, the language of Example 3.1 is not XML.

We now give a more syntactic characterization of XML-languages. For this, we define the set of *contexts* in L of a word w as the set $C_L(w)$ of pairs of words (x, y) such that $xwy \in L$.

Theorem 4.3. *A language L over $A \cup \bar{A}$ is an XML-language if and only if*

- (i) $L \subset D_\alpha$ for some $\alpha \in A$,
- (ii) for all $a \in A$ and $w, w' \in F_a(L)$, one has $C_L(w) = C_L(w')$,
- (iii) the set $S_a(L)$ is regular for all $a \in A$.

Before giving the proof, let us compute one example.

Example 4.4. Consider the language L generated by the grammar

$$\begin{aligned} S &\rightarrow aTT\bar{a} \\ T &\rightarrow aTT\bar{a} \mid b\bar{b} \end{aligned}$$

with axiom S . This grammar is not XML. Clearly, $L \subset D_a$. Also, $F_a(L) = L$. There is a unique set $C_L(w)$ for all $w \in L$, because at any place in a word in L , a factor w in L can be replaced by another factor w' in L . Finally, $S_a(L) = (a \cup b)^2$ and $S_b(L) = \varepsilon$. The theorem claims that there is an XML-grammar generating L .

Proof. We write F_a , S_a and $C(w)$, with the language L understood. We first show that the conditions are sufficient.

Let G be the XML-grammar defined by the family S_a and with axiom X_α . We prove first $L_G(X_a) = F_a$ for $a \in A$. We let the proof of the inclusion $F_a \subset L_G(X_a)$ to the reader. Next, we prove the inclusion $F_a \supset L_G(X_a)$ by induction on the derivation length k . Assume $X_a \xrightarrow{k} w$. Then $w = au\bar{a}$ for some word u . If $k = 1$, then the empty word is in S_a , which means that $a\bar{a}$ is in F_a . If $k > 1$, then the derivation factorizes in

$$X_a \rightarrow aX_{a_1} \cdots X_{a_n} \bar{a} \xrightarrow{k-1} au\bar{a}$$

for some production $X_a \rightarrow aX_{a_1} \cdots X_{a_n} \bar{a}$. Thus there is a factorization $u = u_1 \cdots u_n$ such that $u_i \in L_G(X_{a_i})$ for $i = 1, \dots, n$. By induction, $u_i \in F_{a_i}$ for $i = 1, \dots, n$. Moreover, the word $a_1 \cdots a_n$ is in the surface S_a . This means that there exist words u'_i in F_{a_i} such that the word $w' = au'_1 \cdots u'_n \bar{a}$ is in F_a . Let g, d be two words such that $gw'd$ is in the language L . Then the pair $(ga, u'_2 \cdots u'_n \bar{a}d)$ is a context for the word u'_1 . By (ii), it is also a context for u_1 . Thus $au_1 u'_2 \cdots u'_n \bar{a}$ is in F_a . Proceeding in this way, one strips off all primes in the u 's, and eventually $au_1 u_2 \cdots u_n \bar{a}$ is in F_a . Thus w is in F_a . This proves the inclusion and therefore

the equality. Finally, by Corollary 3.4, one has $L_G(X_\alpha) = L$, and thus we have shown that the conditions are sufficient.

We now show that the conditions are necessary. Let G be an XML-grammar generating L , with productions $X_a \rightarrow aR_a\bar{a}$ and axiom X_α . Clearly, L is a subset of D_α . Next, consider words $w, w' \in F_a$ for some letter a , and let (g, d) be a context for w . Thus $gwd \in L$. By Lemma 3.3, we know that $F_a = L_G(X_a)$. Thus, there exist derivations $X_a \xrightarrow{*} w$ and $X_a \xrightarrow{*} w'$. Substituting the second to the first in

$$X_\alpha \xrightarrow{*} gX_ad \xrightarrow{*} gwd$$

shows that (g, d) is also a context for w' . This proves condition (ii).

Finally, since R_a is a regular set, the set S_a is also regular. $\square$

Example 4.5. Consider the language L of Example 4.4. The construction of the proof of the theorem gives the XML-grammar

$$\begin{aligned} X_a &\rightarrow a(X_a|X_b)(X_a|X_b)\bar{a} \\ X_b &\rightarrow b\bar{b} \end{aligned}$$

Example 4.6. The language

$$\{a(b\bar{b})^n(c\bar{c})^n\bar{a} \mid n \geq 1\}$$

already given above is not XML since the surface of a is the nonregular set $S_a = \{b^n c^n \mid n \geq 1\}$. This is the formalization of the example given in the introduction, if the tag b means bold paragraphs, and the tag c means italic paragraphs.

Example 4.7. Consider again the language

$$L = \{ab^{2n}\bar{b}^{2n}\bar{a} \mid n \geq 1\}$$

of Example 3.1. We compute some contexts. First $C_L(b\bar{b}) = \{(ab^{2n-1}, a\bar{b}^{2n-1}\bar{a}) \mid n \geq 1\}$. Next $C_L(b^2\bar{b}^2) = \{(ab^{2n}, a\bar{b}^{2n}\bar{a}) \mid n \geq 0\}$. Thus there are factors with distinct contexts. This shows again that the language is not XML.

Finally, we give an example showing that XML-languages are not closed under union nor difference.

Example 4.8. Consider the sets $cL\bar{c}$ and $cM\bar{c}$, where $L = D_{\{a,b\}}^*$ is the set of products of Dyck primes over $\{a, b\}$, and $M = D_{\{a,d\}}^*$ is the set of products of Dyck primes over $\{a, d\}$. Each of these two languages is XML. However, the union $H = L \cup M$ is not. Indeed, the words $cabb\bar{a}\bar{c}$ and $ca\bar{a}dd\bar{c}$ are both in H . The pair $(c, dd\bar{c})$ is in the context of $a\bar{a}$, so it has to be in the context of $ab\bar{b}\bar{a}$, but the word $cabb\bar{a}dd\bar{c}$ is not in H . Since XML-languages are closed under intersection, this proves that they are not closed under difference.

5 Decision Problems

As usual, we assume that languages are given in an effective way, in general by a grammar or an XML-grammar, according to the assumption of the statement.

Some properties of XML-languages, such as inclusion or equality (Proposition 3.10) are easily decidable because they reduce to decidable properties of regular sets. The problem is different if one asks whether a context-free grammar generates an XML-language. We have already seen in Example 4.4 that there exist context-free grammars that generate XML-languages without being XML-grammars. The following proposition and its proof are an extension of a result proved by Knuth [3] for parenthesis grammars.

Proposition 5.1. *Given a context-free language L over the alphabet $A \cup \bar{A}$, it is decidable whether $L \subset D_A$.*

In the same paper, Knuth proves also that it is decidable whether a context-free grammar generates a parenthesis language. In contrast to this decidability result, we have.

Proposition 5.2. *It is undecidable whether a context-free language is an XML-language. It is undecidable whether the surfaces of a context-free language are regular.*

This proposition is one reason to consider finite surfaces. Also, the associated XML-grammar is then a context-free grammar in the strict sense, that is with a finite number of productions for each nonterminal. Finally, XML-grammars with finite surfaces are very close to families of grammars that were studied a long time ago. They will be described in the historical note below. The main result is the following.

Theorem 5.3. *Given a context-free language L that is a subset of a set of Dyck primes, it is decidable whether L has all its surfaces finite.*

Conclusion. Given an ordinary context-free grammar G over $A \cup \bar{A}$, we have seen that it is decidable whether $L(G) \subset D_a$ for some letter $a \in A$. If the inclusion holds, we have shown that it is decidable whether $L(G)$ has finite surfaces. If this holds, one may proceed further. A generalization of an argument of Knuth [3] allows to build a balanced grammar G' such that $L(G) = L(G')$. Finally, using this grammar and extending a result of McNaughton [5], one can decide whether the language $L(G')$ is an XML-language.

6 Historical Note

There exist several families of context-free grammars related to XML-grammars that have been studied in the past. In the sequel, the alphabet of nonterminals is denoted by V .

Parenthesis grammars. These grammars have been studied by McNaughton [5] and by Knuth [3]. A *parenthesis grammar* is a grammar with terminal alphabet $T = B \cup \{a, \bar{a}\}$, and where every production is of the form $X \longrightarrow am\bar{a}$, with $m \in (B \cup V)^*$. A parenthesis grammar is *pure* if $B = \emptyset$. In a parenthesis grammar, every derivation step is marked, but there is only one kind of tag.

Bracketed grammars. These were investigated by Ginsburg and Harrison in [1]. The terminal alphabet is of the form $T = A \cup \bar{B} \cup C$ and productions are of the form $X \longrightarrow amb$, with $m \in (V \cup C)^*$. Moreover, there is a bijection between A and the set of productions. Thus, in a bracketed grammar, every derivation step is marked, and the opening tag identifies the production that is applied (whereas in an XML-grammar they only give the nonterminal).

Very simple grammars. These grammars were introduced in [4], and studied in depth later on. Here, the productions are of the form $X \longrightarrow am$, with $a \in A$ and $m \in V^*$. In a simple grammar, the pair (a, m) determines the production, and in a very simple grammar, there is only one production for each a in A .

Chomsky-Schützenberger grammars. These grammars are used in the proof of the Chomsky-Schützenberger theorem (see e. g. [2]), even if they were never studied for their own. Here the terminal alphabet is of the form $T = A \cup \bar{A} \cup B$, and the productions are of the form $X \longrightarrow am\bar{a}$. Again, there is only one production for each letter $a \in A$.

XML-grammars differ from all these grammars by the fact that the set of productions is not necessarily finite, but regular. However, one could consider a common generalization, by introducing *balanced grammars*. In such a grammar, the terminal alphabet is $T = A \cup \bar{A} \cup B$, and productions are of the form $X \longrightarrow am\bar{a}$, with $m \in (V \cup B)^*$. Each of the parenthesis grammars, bracketed grammars, Chomsky-Schützenberger grammars are balanced. If $B = \emptyset$, such a *pure* grammar covers XML-grammars with finite surfaces. If the set of productions of each nonterminal is allowed to be regular, one gets a new family of grammars with interesting properties.

References

1. S. Ginsburg and M. A. Harrison. Bracketed context-free languages. *J. Comput. Syst. Sci.*, 1:1–23, 1967.
2. Michael A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Mass., 1978.
3. D. E. Knuth. A characterization of parenthesis languages. *Inform. Control*, 11:269–289, 1967.
4. A. J. Korenjak and J. E. Hopcroft. Simple deterministic grammars. In *7th Switching and Automata Theory*, pages 36–46, 1966.
5. R. McNaughton. Parenthesis grammars. *J. Assoc. Mach. Comput.*, 14:490–500, 1967.
6. W3C Recommendation REC-xml-19980210. *Extensible Markup Language (XML) 1.0*, 10 February 1998. <http://www.w3.org/TR/REC-XML>.
7. W3C Working Draft. *Canonical XML Version 1.0*, 15 November 1999. <http://www.w3.org/TR/xml-c14n>.

Simplifying Flow Networks

Therese C. Biedl^{*}, Broňa Brejová^{**}, and Tomáš Vinar^{***}

Department of Computer Science, University of Waterloo
{biedl,bbrejova,tvinar}@uwaterloo.ca

Abstract. Maximum flow problems appear in many practical applications. In this paper, we study how to simplify a given directed flow network by finding edges that can be removed without changing the value of the maximum flow. We give a number of approaches which are increasingly more complex and more time-consuming, but in exchange they remove more and more edges from the network.

1 Background

The problem of finding a maximum flow in a network is widely studied in literature and has many practical applications (see for example [1]). In particular we are given a network (G, s, t, c) where $G = (V, E)$ is a directed graph with n vertices and m edges, s and t are two vertices (called *source* and *sink* respectively) and $c : E \rightarrow R^+$ is a function that defines capacities of the edges. The problem is to find a maximum flow from s to t that satisfies capacity constraints on the edges.

Some graphs contain vertices and edges that are not relevant to a maximum flow from s to t . For example, vertices that are unreachable from s can be deleted from the graph without changing the value of maximum flow. In this article we study how to detect such useless vertices and edges.

The study of such a problem is motivated by two factors. First, some graphs may contain a considerable amount of useless edges and by removing them we decrease the size of the input for maximum flow algorithms. Also, the optimization of the network itself may be sometimes desired. Second, some maximum flow algorithms may require that the network does not contain useless edges to achieve better time complexity (see [9] for an example of such algorithm). The precise definition of a useful edge is as follows:

Definition 1. *We call an edge e useful if for some assignment of capacities, every maximum flow uses edge e . If e is not useful, then we call it useless.*

Note in particular that the definition of a useful edge depends only on the structure of the network and the placement of the source and sink, but not on

^{*} Supported by NSERC Research Grant.

^{**} Supported by NSERC Research Grant OGP0046506 and ICR Doctoral Scholarship.

^{***} Supported by NSERC Research Grant OGP0046506.

the actual capacities in the network. Thus, a change in capacities would not affect the usefulness of an edge.¹

The above definition of a useful edge is hard to verify. As a first step, we therefore develop equivalent characterization which uses directed paths.²

Lemma 1. *The following conditions are equivalent:*

1. *Edge $e = (v, w)$ is useful.*
2. *There exists a simple path P from s to t that uses e .*
3. *There exist vertex-disjoint paths P_s from s to v and P_t from w to t .*

Proof Sketch. Assume that (1) holds. Let c be capacities such that any maximum flow contains e . By the flow decomposition theorem there exists a maximum flow f that can be decomposed into flows along simple paths from s to t . Since f uses e , (2) holds.

To prove that (2) implies (1) we set all capacities on P to 1, all other capacities to 0. The equivalence of (2) and (3) is obvious by adding/deleting edge e . $\square$

This implies that testing whether an edge is useless is NP-complete, by reducing it to the problem of finding two disjoint paths between two given sources and sinks [3]. Thus we relax the problem in two ways:

- Find edges that are clearly useless, without guaranteeing that all useless edges will be found. In particular, in Section 2 we will give a number of conditions under which an edge is useless. For each of them we give an algorithm how to find all such edges.
- Restrict our attention to planar graphs without clockwise cycles. We present in Section 3 an $O(n)$ time algorithm to test whether a given edge is useless.

We conclude in Section 4 with open problems. For space reasons many details have been left out; a full-length version can be found in [2].

2 Finding Some Edges That Are Useless

As proved above, an edge (v, w) is useless if and only if there are two vertex-disjoint paths P_s from s to v and P_t from w to t . Unfortunately, it is NP-complete to test whether an edge is useless. We will therefore relax this characterization of “useless” in a variety of ways as follows:

- We say that an edge $e = (v, w)$ is *s-reachable* if there is a path from s to v , and *s-unreachable* otherwise. We say that an edge $e = (v, w)$ is *t-reachable* if there is a path from w to t , and *t-unreachable* otherwise.

¹ It would seem a natural approach to extend the definition of useful edges in a way that takes capacities into account. However such an extension is not straightforward, and we will not pursue such an alternative definition.

² In this paper, we will never consider paths that are not directed, and will therefore drop “directed” from now on.

- We say that an edge $e = (v, w)$ is *s-useful* if there is a *simple* path from s to w that ends in e , and *s-useless* otherwise. We say that an edge $e = (v, w)$ is *t-useful* if there is a simple path from v to t that begins with e , and *t-useless* otherwise.
- We say that an edge $e = (v, w)$ is *s-and-t-useful* if it is both *s-useful* and *t-useful*, and *s-or-t-useless* otherwise.

Notice that each edge that is “useless” according to some of these definitions, is clearly useless according to Definition 1. Unfortunately, even an *s-and-t-useful* edge is not necessarily useful.

Edges that are *s-reachable* can be easily detected using a directed search from s in $O(m + n)$ time (similarly we can also detect all *t-reachable* edges). However, the other introduced concepts cannot be handled so easily. We study for each of these concepts how to detect such edges in the following subsections.

2.1 s-Useful Edges

Recall that an edge e is *s-useful* if there exists a simple path starting at s and using e , and *s-useless* otherwise. Note that not every *s-reachable* edge (v, w) is *s-useful*, because it might be that all paths from s to v must go through w . In particular, an edge $e = (v, w)$ is *s-useful* if and only if there is a path from s to v that does not pass through w . This observation can be used to determine *s-useless* edges in $O(mn)$ time (perform n times a depth-first search starting in s , each time with one vertex removed from the graph). We will improve on the time complexity of this simple algorithm, and present an algorithm to find all *s-useless* edges in $O(m \log n)$ time.

Our algorithm works by doing three traversals of the graph. The *first traversal* is simply a depth-first search starting in s . Edges not examined during this search are *s-unreachable*, hence *s-useless*. We will not consider such edges in the following.

Let T be the depth-first search tree computed in the first traversal. Let $v_1, v_2, \dots, v_n$ be the vertices in the order in which they were discovered during the depth-first search. Let the *DFS-number* $\text{num}(v)$ be such that $\text{num}(v_i) = i$.

A depth-first search of a directed graph divides the edges into the following categories: *tree edges* (included in the DFS-tree), *back edges* (leading to an ancestor), *forward edges* (leading to a descendant that is not a child), and *cross edges* (leading to a vertex in a different subtree). By properties of a depth-first search, cross edges always lead to a vertex with a smaller DFS-number (see for example [1]).

In the *second traversal* we compute for each vertex v a value $\text{detour}(v)$, which roughly describes from how far above v we can reach v without using vertices in the tree inbetween. The precise definition is as follows (see Fig. 1a for an illustration).

Definition 2. If u and v are two vertices, and u is an ancestor³ of v in the DFS-tree, then denote by $T(u, v)$ the vertices in the path between u and v in the DFS-tree (excluding the endpoints), and let $T(u, v]$ be $T(u, v) \cup \{v\}$.

³ A vertex v is considered to be an ancestor of itself.

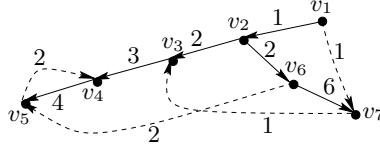


Fig. 1. The DFS-tree is shown with solid lines. Every edge e is labeled by $\text{detour}(e)$.

Assume that vertex u is an ancestor of vertex v . A path from u to v will be called a detour if it does not use any vertices in $T(u, v)$.⁴ For an edge $e = (w, v)$, we denote by $\text{detour}(e)$ the minimum value i for which there exists a detour from v_i to v that uses edge e as its last edge. For a vertex v , we denote by $\text{detour}(v)$ the minimum value i for which there exists a detour from v_i to v . In particular therefore, $\text{detour}(v)$ is the minimum of $\text{detour}(e)$ over all incoming edges e of v .

In the *third traversal* we finally compute for each edge whether it is s -useless or not. The second and third traversal are non-trivial and will be explained in more detail below.

The Second Traversal. We compute detours by scanning the vertices in reverse DFS-order. To compute $\text{detour}(v)$ we need to compute $\text{detour}(e)$ for all edges e incoming to v . The following lemma formulates the core observation used in the algorithm.

Lemma 2. Let $e = (w, v)$ be an edge. If e is a tree edge or a forward edge then $\text{detour}(e) = \text{num}(w)$. If e is a cross edge or a backward edge and a is the nearest common ancestor of v and w ($a = v$ if e is a back edge) then

$$\text{detour}(e) = \min_{u \in T(a, w]} \text{detour}(u). \quad (1)$$

Proof Sketch. Let e be a tree edge or a forward edge. Obviously e is itself a detour and therefore $\text{detour}(e) = \text{num}(w)$.

Assume that e is a cross or backward edge. Let $d = \min_{u \in T(a, w]} \text{detour}(u)$, let u be a vertex in $T(a, w)$ that achieves the minimum, and let Q be a detour from v_d to u (see the left picture of Fig. 2). It is possible to show that v_d is an ancestor of a and that $Q - \{v_d\}$ does not contain a vertex b with $\text{num}(b) < \text{num}(u)$. The latter claim follows from the properties of DFS-tree. Let P be a path from v_d to v starting with Q from v_d to u , then following tree-edges from u to w , and ending with edge e . Since all vertices $b \in T(v_d, v]$ have $\text{num}(b) < \text{num}(w)$, path P is indeed a detour for e . This gives us $\text{detour}(e) \leq d$.

On the other hand, let P be the detour that ends with edge $e = (w, v)$ that starts in a vertex x such that $\text{num}(x) = \text{detour}(e)$ (see the right picture of Fig. 2). Observe that x is an ancestor of a . Let y be the first vertex on P that belongs to $T(a, w]$. Then the part of P from x to y is a detour for y and therefore $d \leq \text{detour}(y) \leq \text{detour}(e)$. $\square$

⁴ We allow $u = v$, in case of which any path from v to v is a detour.

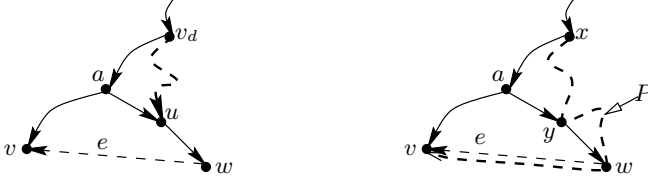


Fig. 2. Computation of $\text{detour}(e)$ for a cross edge. The case of a back edge is similar.

Using this lemma, we can compute the detour-values during the second traversal. We use the dynamic trees (DT) data structure of [8], which we initialize to set of isolated vertices $\{v_1, v_2, \dots, v_n\}$ in $O(n)$ time.

Now we process the vertices in reverse DFS-order (i.e., from v_n to v_1). After we have computed $\text{detour}(v_j)$ ($j > 1$), we update DT by linking v_j to its parent in T with an edge of weight $\text{detour}(v_j)$. This can be done in $O(\log n)$ time [8]. Therefore DT contains in each step a subset of edges in T so that after processing v_j it contains all tree edges with at least one endpoint from $\{v_j, \dots, v_n\}$. In particular, this means that each vertex v_i for $i < j$ is a root of a tree in DT.

To compute $\text{detour}(v_j)$, we compute $\text{detour}(e)$ for all incoming edges e of v_j and take the minimum. The value $\text{detour}(e)$ is computed using Lemma 2. The value for tree and forward edges can be determined in straightforward way in $O(1)$ time. For back and cross edges we need to compute $\min_{u \in T(a, w]} \text{detour}(u)$, where a is the nearest common ancestor of v_j and w .

Observe that $\text{num}(a) \leq \text{num}(v_j)$ and therefore a is the root of the tree in DT containing w . Therefore $\text{detour}(e)$ is the minimum value of an edge in DT on the path from w to the root of the tree containing w in DT (this root is a). Such a minimum can be computed in $O(\log n)$ time in dynamic trees [8]. Hence we can determine $\text{detour}(e)$ in $O(\log n)$ time. Altogether, this traversal takes $O(m \log n)$ time.

The Third Traversal. In the third traversal, we determine for each edge $e = (v, w)$ whether it is s -useful. If e is a tree edge, a forward edge or a cross edge, then the path from s to v in tree T followed by e is simple and e is s -useful. Thus we only consider back edges, and need the following lemma, whose proof is omitted for space reasons.

Lemma 3. *Let $e = (v, w)$ be a back edge. Then e is s -useful if and only if for some $u \in T(w, v]$ we have $\text{detour}(u) < \text{num}(w)$.*

To actually determine the s -useful edges, we use interval trees [6], where the endpoints of intervals are in $\{1, 2, \dots, n\}$. The interval tree is initialized as an empty tree in $O(n)$ time. We perform yet another depth-first search (with exactly the same order of visiting vertices). When reaching vertex v , the interval tree contains intervals of the form $(\text{detour}(u), \text{num}(u))$ for all ancestors u of v . We add interval $(\text{detour}(v), \text{num}(v))$ into the tree. When we retreat from v , we delete $(\text{detour}(v), \text{num}(v))$ from the interval tree. Inserting and deleting an interval takes $O(\log n)$ time per vertex [6].

Assume now we are currently processing vertex v . For each back edge (v, w) we want to check whether there is a vertex $x \in T(w, v]$ with $\text{detour}(x) < \text{num}(w)$. But this is the case if and only if $\text{detour}(x) < \text{num}(w) < \text{num}(x)$ and x is an ancestor of v . But the interval $(\text{detour}(x), \text{num}(x))$ is stored in the interval tree for all ancestors of v ; thus edge $e = (v, w)$ is s -useful if and only if there is an interval (a, b) in the tree with $a < \text{num}(w) < b$. This can be tested in $O(\log n)$ time per back edge [6], and hence $O(m \log n)$ time total.

Theorem 1. *All s -useless edges can be found in $O(m \log n)$ time.*

2.2 s -and- t -Useful Edges

Recall that an edge e is s -or- t -useless if it is s -useless or t -useless. In the previous section we have shown how to find all s -useless edges in $O(m \log n)$ time. The algorithm can be adapted easily to find all t -useless edges, simply by reversing the direction of all edges in the graph and using vertex t instead of s . Therefore the following lemma holds.

Lemma 4. *All s -or- t -useless edges of a graph can be found in $O(m \log n)$ time.*

Surprisingly so, this lemma does not solve the problem how to obtain a graph without s -or- t -useless edges, because removing t -useless edges may introduce new s -useless edges and vice versa.

To obtain a graph without s -or- t -useless edges we therefore need to repeat the procedure of detecting and removing s -or- t -useless edges until no such edges are found. Clearly, each iteration removes at least one edge, which gives $O(m)$ iterations and $O(m^2 \log n)$ time. Unfortunately, $\Omega(n)$ iterations may also be needed (see Fig. 3).

We conjecture that $O(n)$ iterations always suffice, but this remains an open problem. One would expect that in practice the number of iterations is small, and hence this algorithm would terminate quickly. Another open problem is to develop a more efficient algorithm to find all edges that need to be removed to obtain a graph without s -or- t -useless edges.

3 Finding Useless Edges in Planar Graphs

Recall that an edge $e = (v, w)$ is *useful* if there are vertex-disjoint paths P_s from s to v and P_t from w to t . As mentioned in the introduction, it is NP-complete to test whether a given edge is useful in a general graph. By a result of Schrijver [7] the problem is solvable in polynomial time in planar graphs. However, he did not study the precise time complexity of his algorithm, which seems to be high.

We present an algorithm finding in $O(n^2)$ time all useless edges in a planar graph. Our algorithm assumes that a fixed planar embedding is given such that t is on the outerface and the graph contains no clockwise cycles. Not all planar graphs have such an embedding, but for fixed capacities and fixed planar embedding with t on the outerface, it is possible, in $O(n \log n)$ time, to modify graph so

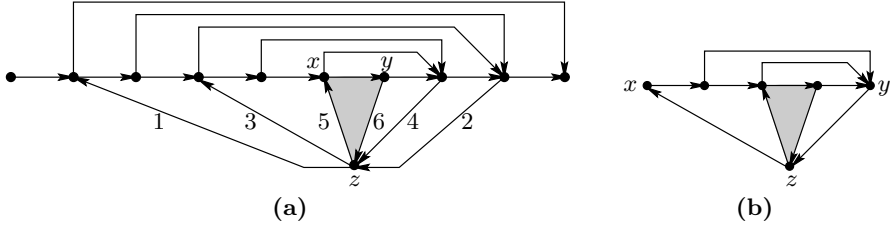


Fig. 3. Construction of a graph with $3k + 2$ vertices and $7k - 1$ edges that requires $\Omega(n)$ iterations. In the first iteration, edge 1 is detected as s -useless and deleted, which makes edge 2 t -useless. Deleting edge 2 makes edge 3 s -useless. This continues, and one can show that we need $2k = \Omega(n)$ iterations. (a) The graph for $k = 3$. (b) The graph for higher k can be obtained by repeatedly replacing the shaded triangle with the depicted subgraph.

that the maximum flow is not changed and all clockwise cycles are removed (see [5]). Therefore in general our algorithm finds for any planar network a planar network with the same maximum flow not containing useless edges.

The algorithm is unfortunately too slow to be relevant for maximum flow problems.⁵ Still we believe that the problem is interesting enough in its own right to be worth studying.

The following notation will be useful later: Assume that $P = w_1, \dots, w_k$ is a path. We denote by $P[w_i, w_j]$ ($i \leq j$) the sub-path of P between vertices w_i and w_j . We denote by $P(w_i, w_j)$ ($i < j$) the sub-path of P between vertices w_i and w_j excluding the endpoints. Note that this sub-path might consist of just one edge (if $i = j - 1$).

The crucial ingredient to our algorithm is the following observation, which holds even if the graph is not planar.

Lemma 5. *Let $e = (v, w)$ be an edge, let P_s be a simple path from s to v and let P_t be a simple path from w to t . If P_s and P_t have vertices in common, then there exists a simple directed cycle C that contains e such that some vertex $x \in C$ belongs to both P_s and P_t .*

Proof Sketch. Let x be the last vertex of P_s that belongs to P_t . Then $P_s[x, v] \cup (v, w) \cup P_t[w, x]$ forms the desired cycle. $\square$

We will use the contrapositive of this observation: Assume that we have paths P_s and P_t , and they do not have a common vertex on any directed cycle containing e . Then P_s and P_t are vertex-disjoint.

Using planarity we will show that we need to examine only the “rightmost” cycle containing e . To define this cycle, we use the concept of a *right-first search*: perform a depth-first search, and when choosing the outgoing edge for the next step, take the counter-clockwise next edge after the edge from which we arrived.

The *rightmost cycle containing edge $e = (v, w)$* consists of edge e and the path from w to v in the DFS-tree constructed by the right-first search starting in w with the first outgoing edge (in counter-clockwise order) after e .

⁵ A maximum flow problem can be solved in subquadratic time for planar graphs. See for example the $O(n^{3/2} \log n)$ time algorithm by Johnson and Venkatesan [4].

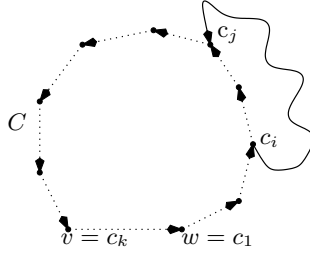


Fig. 4. A path from c_i to c_j , $i < j$ must either be inside C or contain other vertices of C . Otherwise C would not be the rightmost cycle containing (v, w) .

Observe that the rightmost cycle containing e is not defined if and only if there exists no directed cycle containing e . However, in this case by Lemma 5 edge e is useful if and only if it is s -reachable and t -reachable, which can be tested easily. So we assume for the remainder that the rightmost cycle C is well-defined, and it must be counter-clockwise because by assumption there are no clockwise cycles in the graph.

Enumerate C , starting at w , as $w = c_1, c_2, \dots, c_k = v$. Cycle C defines a closed Jordan curve, and as such has an inside and an outside. We say that a vertex is *inside* (*outside*) C if it does not belong to C and is inside (outside) the Jordan curve defined by C . The following observation will be helpful later:

Lemma 6. *For any $1 \leq i < j \leq k$, any simple path P from c_i to c_j must either contain a vertex $\neq c_i, c_j$ in C or $P(c_i, c_j)$ must be inside C .*

Proof Sketch. If there were a path that satisfies neither condition, then using it we could find a cycle containing e that is “more right” than C (see Fig. 4). $\square$

We may assume that s has no incoming and t has no outgoing edges (such edges would be useless). Therefore $s, t \notin C$. On the other hand, both v and w belong to C . Hence, for any path from s to v there exists a first vertex $x \neq s$ on C ; we mark vertex x as an *entrance*. Similarly, for any path from w to t there exists a last vertex $y \neq t$ on C ; we mark y as an *exit*.

The following two lemmas show that we can determine whether e is useful from the markings as entrances and exits on C alone.

Lemma 7. *If there is an exit c_i and an entrance c_j with $1 \leq i < j \leq k$, then e is useful.*

Proof Sketch. Let P'_s be a path from s to v that marked c_j as an entrance and let P'_t be a path from w to t that marked c_i as exit. Let $P_s = P'_s[s, c_j] \cup \{c_j, \dots, c_k = v\}$ and let $P_t = \{w = c_1, \dots, c_i\} \cup P'_t[c_i, t]$. Clearly, these are paths from s to v and from w to t , and using Lemma 6, one can argue that they are vertex-disjoint. See also Fig. 5a. $\square$

Now we show that the reverse of Lemma 7 also holds.

Lemma 8. *Let $j_{\max}$ be maximal such that $c_{j_{\max}}$ is an entrance, and let $i_{\min}$ be minimal such that $c_{i_{\min}}$ is an exit. If $j_{\max} \leq i_{\min}$, then e is useless.*

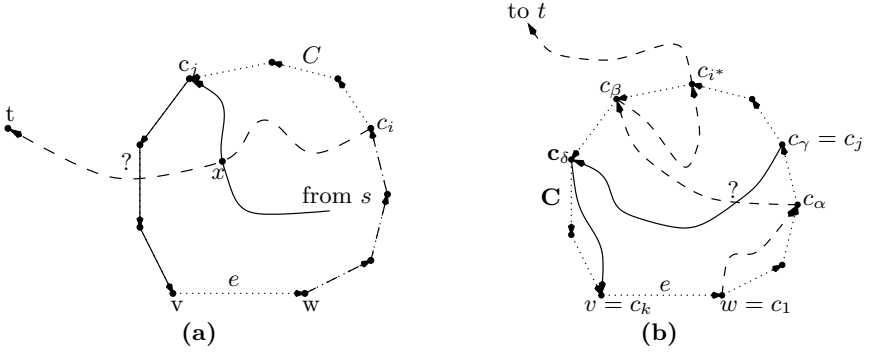


Fig. 5. (a) Paths P_s (solid) and P_t (dashed) are vertex disjoint, for if they intersect, it would have to be at a vertex x inside C , which contradicts planarity and the fact that t is outside C . (b) Illustration of the definition of $i^*, j^*, j, \alpha, \beta, \gamma$ and δ . Path P_s has solid, path P_t has dashed lines.

Proof Sketch. Assume to the contrary that e is useful, and let P_s and P_t be the vertex-disjoint paths from s to v and from w to t , respectively. Let c_{j^*} be the entrance defined by P_s , and let c_{i^*} be the exit defined by P_t . By assumption and vertex-disjointness, we have $j^* < i^*$.

Let $j > 1$ be minimal such that $c_j \in P_s$. Let c_β be the first vertex of P_t that belongs to C and satisfies $j < \beta$, and let c_α be the last vertex of $P_t(v, c_\beta)$ that belongs to C . Similarly, let c_δ be the first vertex of P_s after c_j that belongs to C with $\delta \notin [\alpha, \beta]$. Let c_γ be the last vertex of $P_s(s, c_\delta)$ that belongs to C .

By careful observation it is possible to show that $j, \alpha, \beta, \gamma, \delta$ are well-defined and $\alpha < \gamma < \beta < \delta$ (see Fig. 5b).

Now $C \cup P_t[c_\alpha, c_\beta] \cup P_s[c_\gamma, c_\delta]$ forms a subdivision of K_4 . But by Lemma 6, $P_t(c_\alpha, c_\beta)$ and $P_s(c_\gamma, c_\delta)$ are both inside C because they contain no other vertex of C . Therefore, C is the outer-face of this subdivided K_4 , which implies that K_4 is outer-planar. This is a contradiction. $\square$

Computing C and marking exits and entrances can be done with directed searches in $O(n + m)$ time, so the following result holds:

Theorem 2. *Testing whether e is useless in a planar graph with t on the outerface and without clockwise cycles can be done in $O(n)$ time.*

4 Conclusion

In this paper, we studied how to simplify flow networks by detecting and deleting edges that are useless, i.e., that can be deleted without changing the maximum flow. Detecting all such edges is NP-complete. We first studied how to detect at least some useless edges. More precisely, we defined when an edge is s -useless, and showed how to find all s -useless edges in $O(m \log n)$ time.

We also studied other types of useless edges, in particular s - and t -useless edges, and useless edges in planar graphs without clockwise cycles. While for

both types we give algorithms to find such edges in polynomial time (more precisely, $O(m^2 \log n)$ and $O(n^2)$), these results are not completely satisfactory, because one would want to find such edges in time less than what is needed to compute a maximum flow per se. Thus, we leave the following open problems:

- We gave an example of a graph where computing s -or- t -useless edges takes $\Omega(n)$ rounds, hence our technique cannot be better than $O(mn \log n)$ time, which is too slow. Is there a “direct” approach to detect a subgraph without s -or- t -useless edges that has the same maximum flow?
- Can our insight into the structure of useless edges in planar graphs be used to detect *all* useless edges in a planar graph in time $O(n \log n)$ or even $O(n)$?
- Currently, our algorithm for useless edges in planar graphs works only if the planar graph has no clockwise cycles. Is there an efficient algorithm which works without this assumption?
- The problem of finding maximum flow in directed planar graphs in $O(n \log n)$ time seemed to be solved by [9]. However this algorithm needs s -or- t useless edges to be removed in preprocessing and the author does not provide a solution for this problem. Therefore we consider the following question still open: Is it possible to find a maximum flow in directed planar graphs in $O(n \log n)$ time?

References

1. Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.
2. Therese Biedl, Broňa Brejová, and Tomáš Vinař. Simplifying flow networks. Technical Report CS-2000-07, Department of Computer Science, University of Waterloo, 2000.
3. Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, February 1980.
4. Donald B. Johnson and Shankar M. Venkatesan. Using divide and conquer to find flows in directed planar networks in $O(n^{3/2} \log n)$ time. In *Proceedings of the 20th Annual Allerton Conference on Communication, Control, and Computing*, pages 898–905, University of Illinois, Urbana-Champaign, 1982.
5. Samir Khuller, Joseph Naor, and Philip Klein. The lattice structure of flow in planar graphs. *SIAM Journal on Discrete Mathematics*, 6(3):477–490, August 1993.
6. Franco P. Preparata and Micheal I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.
7. Alexander Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM Journal of Computing*, 23(4):780–788, August 1994.
8. Daniel D. Sleator and Robert E. Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, June 1983.
9. Karsten Weihe. Maximum (s, t) -flows in planar networks in $\mathcal{O}(|V| \log |V|)$ time. *Journal of Computer and System Sciences*, 55(3):454–475, December 1997.

Balanced k -Colorings

Therese C. Biedl, Eowyn Čenek, Timothy M. Chan, Erik D. Demaine,
Martin L. Demaine, Rudolf Fleischer, and Ming-Wei Wang

Department of Computer Science, University of Waterloo
{biedl,ewcenek,tmchan,eddemaine,mldemaine,rudolf,m2wang}@uwaterloo.ca

Abstract. While discrepancy theory is normally only studied in the context of 2-colorings, we explore the problem of k -coloring, for $k \geq 2$, a set of vertices to minimize imbalance among a family of subsets of vertices. The *imbalance* is the maximum, over all subsets in the family, of the largest difference between the size of any two color classes in that subset. The *discrepancy* is the minimum possible imbalance. We show that the discrepancy is always at most $4d - 3$, where d (the “dimension”) is the maximum number of subsets containing a common vertex. For 2-colorings, the bound on the discrepancy is at most $\max\{2d - 3, 2\}$. Finally, we prove that several restricted versions of computing the discrepancy are NP-complete.

1 Introduction

We begin with some basic notation and terminology. Let $\mathcal{L}$ be a family of nonempty subsets of a finite set P . We call the elements of P *vertices* and the elements of $\mathcal{L}$ *lines*. A vertex $v \in P$ *lies on a line* $\ell \in \mathcal{L}$ if $v \in \ell$. We denote the number of vertices on a line ℓ by $|\ell|$.

One topic in the area of combinatorial discrepancy theory [1,4,6,10] is the study of the minimum possible “imbalance” in a 2-coloring of the vertices. Formally, a 2-coloring is a function χ from the vertices in P to the two colors $-1, +1$ (note that we do not speak about coloring the nodes of a graph such that endpoints have different colors). The *imbalance* of χ is the maximum difference between the size of the two color classes considered for every line, i.e., $\max_{\ell \in \mathcal{L}} |\sum_{v \in \ell} \chi(v)|$. The *discrepancy* is the minimum possible imbalance over all 2-colorings; to avoid confusion, we call this standard notion the 2-color *discrepancy*.

In this paper we consider the following more general setting. A k -coloring of P is a mapping from the vertices in P to the k colors $1, \dots, k$. It is called c -balanced if for any line ℓ and any two colors i, j , $1 \leq i, j \leq k$, we have

$$|\#\{\text{vertices on } \ell \text{ colored } i\} - \#\{\text{vertices on } \ell \text{ colored } j\}| \leq c.$$

We call c the *imbalance* of the coloring; it is a strong measure of additive error relative to the uniform distribution. The k -color *discrepancy* of a family $\mathcal{L}$ is the minimum possible imbalance over all k -colorings.

Ideally, we would hope for *perfectly balanced* colorings, i.e., with imbalance 0, but a necessary condition for the existence of a perfectly balanced coloring is that the number of vertices on each line is a multiple of k . Otherwise, the best we can hope for is an *almost-balanced* coloring which is a coloring with imbalance 1. In general, however, even this is not always possible.

Our work is strongly motivated by two previous results. In the context of 2-colorings, Beck and Fiala [5] gave an upper bound of $2d - 1$ on the discrepancy where the *dimension* d (also often called the *maximum degree*) of a family $\mathcal{L}$ is the maximum number of lines passing through a common vertex.

On the other hand, Akiyama and Urrutia [2] studied k -colorings, for arbitrary k , for lines that form a two-dimensional grid and vertices that are a subset of the intersections. In this geometric setting (with $d = 2$), they showed that there is always an almost-balanced k -coloring. The same result for 2-colorings can also be derived using the algorithm by Šíma for table rounding [9] (where each grid point corresponds to a table entry of $\frac{1}{2}$). A more general table rounding problem with applications in digital image processing has recently been studied by Asano et al. [3]. Since finding a k -coloring of points on lines that form a two-dimensional grid can be reformulated as finding an edge k -coloring of a simple bipartite graph, a slightly weaker result follows from a theorem by Hilton and de Werra [7] who showed that a 2-balanced edge k -coloring exists for any simple graph (not just bipartite graphs). Akiyama and Urrutia also showed that not all configurations of points in higher-dimensional grids have an almost-balanced k -coloring, but asked whether an $O(1)$ -balanced k -coloring might be possible for such grids.

In Section 2, we generalize the results of Beck and Fiala [5] and Akiyama and Urrutia [2] to k -colorings of an arbitrary family of lines. In particular, we settle the open questions posed by Akiyama and Urrutia within a constant factor. Specifically, our most general result states that the k -color discrepancy is at most $4d - 3$. Note that this bound is independent of the number of colors.

We can tighten the bound by 1 in the case that the number of vertices on each line is a multiple of k . For 2-colorings we can tighten the bound further to $\max\{2d - 3, 2\}$, improving by an additive constant the results of Beck and Fiala [5]. For $d = 2$, the bound of 2 is tight because the three vertices of a triangle have no almost-balanced 2-coloring. And in the special case of 2-dimensional geometric settings our proof can be strengthened to give the Akiyama and Urrutia result [2], i.e., we can prove there is always an almost-balanced 2-coloring.

In Section 4 we show that a simpler divide-and-conquer algorithm (which is apparently known in the field of discrepancy theory [11] but has never been published to our knowledge) computes k -colorings which are slightly less balanced than the colorings computed by the algorithm in Section 2. Both algorithms can be implemented efficiently in polynomial time.

Finally, in Section 5 we show that for $k \geq 2$ finding almost-balanced k -colorings is NP-complete for line families of dimension at least $\max\{3, k - 1\}$. For $k = 2, 3$, this result even holds for the special case of various geometric settings. We suspect that finding an almost-balanced k -coloring is NP-complete for any $k \geq 2$ and $d \geq 3$.

2 The Balance Theorem

In this section we prove our main theorem which states that any set of vertices on a set of lines of dimension d can be k -colored such that the imbalance on each line is bounded by a constant only depending on the dimension d (and not on k , or on the number of vertices, or on the number of lines).

Theorem 1 (Balance Theorem). *Let $d \geq 2$ and $k \geq 2$. Let $\mathcal{L}$ be a set of lines of dimension d containing a set of vertices P . Then P has a $(4d - 3)$ -balanced k -coloring. The imbalance is at most $4d - 4$ for all lines with a multiple of k many vertices.*

Proof. This proof is an adaptation of the proof given by Beck and Fiala [5] (see also [1,10]) for 2-colorings.

With each vertex $v \in P$ we associate k variables $x_{v,1}, \dots, x_{v,k}$ which will change in time. At all times all $x_{v,i}$ lie in the closed interval $[0, 1]$. Initially, all $x_{v,i}$ are set to $\frac{1}{k}$. If $0 < x_{v,i} < 1$ then $x_{v,i}$ is called *floating*, so initially all variables are floating. If $x_{v,i} = 0$ or $x_{v,i} = 1$ then $x_{v,i}$ is called *fixed*. Once a variable is fixed, it can never change again. Eventually, all variables will be fixed.

At that time, the variables define a k -coloring of the vertices: vertex v is colored with color i if and only if $x_{v,i} = 1$ and $x_{v,j} = 0$ for $j \neq i$. To ensure that there is exactly one i with $x_{v,i} = 1$, we require that at all times

$$\sum_{i=1}^k x_{v,i} = 1 \quad (C_v)$$

for all vertices v . We call the equations (C_v) *color equations*. A color equation is *active* if it contains at least two floating variables; otherwise, it is *inactive*. Note that a color equation contains either zero or at least two floating variables.

For each line ℓ , we want to balance the colors. This can be expressed by k *balance equations* $(E_{\ell,i})$, for $i = 1, \dots, k$:

$$\sum_{v \in \ell} x_{v,i} = \frac{|\ell|}{k} \quad (E_{\ell,i})$$

A balance equation is *active* if it contains at least $2d$ floating variables; otherwise, it is *inactive*.

We can think of the color and balance equations as a system (LP) of linear equations in the variables $x_{v,i}$. Since we cannot always find an integer solution of (LP) (if a line ℓ contains a number of vertices which is not a multiple of k

then there can be no integer solution to the corresponding balance equations), we only require at all times that all active equations are satisfied. We call this subsystem of linear equations (LP_{act}) . Since initially all variables are floating, (LP_{act}) initially consists of all balance equations with at least $2d$ variables.

Now suppose we have a solution of (LP_{act}) at a certain time. Let f be the number of floating variables at this time. Each vertex is contained in at most d lines. Therefore, each variable is contained in at most d balance equations. In particular, each floating variable is contained in at most d active balance equations. On the other hand, each active balance equation contains at least $2d$ floating variables, so the number of active balance equations is at most $\frac{f}{2}$. Moreover, each variable appears in exactly one color equation, whereas each active color equation contains at least two floating variables. Therefore, the number of active color equations is at most $\frac{f}{2}$. If (LP_{act}) is underdetermined, with more variables than equations, then we can move along a line of solutions of (LP_{act}) in Euclidean space. We do so until one of the floating variables reaches 0 or 1; then we stop. This fixes at least one previously floating variable. We continue this procedure until all of the $x_{v,i}$ are fixed, or until (LP_{act}) is not underdetermined. This can only happen if each active balance equation contains exactly $2d$ floating variables and each color equation contains exactly 2 floating variables. Therefore we can round the two floating variables in each color equation to 0 and 1, which changes the value of each balance equation by at most d (we call this the *rounding step*). This yields an imbalance of at most $2d$ (which is less than $4d - 3$ for $d \geq 2$) for the lines corresponding to the active balance equations.

Since the final values of the $x_{v,i}$ still satisfy all the color equations (but not necessarily all the balance equations), we can read off a k -coloring of the vertices. We claim that this k -coloring is $(4d-3)$ -balanced. Consider any balance equation $(E_{\ell,i})$. At the first time it becomes inactive we have $\sum_{v \in \ell} x_{v,i} = \frac{|\ell|}{k}$, and at most $2d-1$ of the $x_{v,i}$ are floating. Later, each of these floating variables can change by less than 1 from that value to its final value. As such, $|\sum_{v \in \ell} x_{v,i} - \frac{|\ell|}{k}| < 2d-1$ for the final values of the $x_{v,i}$. Hence the imbalance is bounded from above by $2 \cdot (2d-2) = 4d-4$ if $|\ell|$ is a multiple of k , and $2 \cdot (2d-1) - 1 = 4d-3$ if $|\ell|$ is not a multiple of k . $\square$

3 2-Colorings

If we want to find balanced 2-colorings we can improve Theorem 1 by approximately a factor of 2.

Theorem 2 (2-Color Balance Theorem). *Let $d \geq 2$. Let $\mathcal{L}$ be a set of lines of dimension d containing a set of vertices P .*

- (a) *If $d = 2$ then P has a 2-balanced 2-coloring.*
- (b) *If $d \geq 3$ then P has a $2d-3$ -balanced 2-coloring. If $d \geq 4$ then the imbalance is at most $2d-4$ for all lines with an even number of vertices.*

Proof. (Sketch) The substitution $x_{v,2} = 1 - x_{v,1}$ eliminates all but the balance equations $(E_{\ell,1})$ from (LP) . Now we call a balance equation *active* if it contains at least d floating variables and proceed as in the proof of Theorem 1. $\square$

Part (a) also follows from the result of Hilton and Werra [7]. In the case of 2-dimensional geometric settings as studied by Akiyama and Urrutia [2] our proof can be adapted to show the existence of an almost-balanced 2-coloring (the rounding step never happens in this case because it can be shown that (LP_{act}) is always underdetermined, even if the number of equations equals the number of variables). We note however that both mentioned previous works give the same discrepancy bound for k -colorings for $k \geq 2$, whereas we only obtain them for 2-colorings.

4 Alternative Approaches

We could try to modify the algorithm given in the proof of Theorem 1 as follows. Instead of computing the colors of all vertices at the same time, we first identify all vertices which should be colored with color 1, then we discard these vertices and identify all vertices which should be colored with color 2, etc. In one step of this iteration, we therefore only need to associate one variable x_v to each vertex v . Finally, when all variables are fixed, the vertices v with $x_v = 1$ belong to one color class; we then iterate on the set of all vertices w with $x_w = 0$. In this approach we do not need color equations, and we have just one *balance equation* (E_ℓ) for each line ℓ :

$$\sum_{v \in \ell} x_v = \alpha |\ell| \quad (E_\ell),$$

where $\alpha = \frac{1}{k}$. A balance equation is *active* if it contains at least $d + 1$ floating variables. As before, we conclude that at any time the system (LP_{act}) of active balance equations is underdetermined, so we can fix at least one floating variable. If all variables are fixed the number of variables with value 1 must be in the open interval $(\alpha|\ell| - d, \alpha|\ell| + d)$ for every line ℓ .

Iterating this procedure yields k color classes. Let e_k be the smallest number such that each color class has size in the interval $(\frac{|\ell|}{k} - e_k, \frac{|\ell|}{k} + e_k)$ along each line ℓ . It is easy to see that we have the recurrence $e_k \leq \max\{d, \frac{d}{k-1} + e_{k-1}\}$, implying $e_k \leq d(1 + \frac{1}{2} + \dots + \frac{1}{k-1})$. This would give us an imbalance of approximately $2d \cdot \ln k$.

To get an imbalance independent of k , we can use divide-and-conquer instead of the iteration just described. If k is even we do one iteration step with α set to $1/2$, i.e., we search for a 2-coloring; we then recursively $(k/2)$ -color the vertices of color 1 and $(k/2)$ -color the remaining vertices. If k is odd we do one iteration step with α set to $\frac{k-1}{2k}$; we then recursively $(k-1)/2$ -color the vertices of color 1 and $(k+1)/2$ -color the remaining vertices. We now have the following recurrence on the error bound e_k : for k even, $e_k \leq \frac{d}{k/2} + e_{k/2}$; for k odd,

$e_k \leq \max\{\frac{d}{(k-1)/2} + e_{(k-1)/2}, \frac{d}{(k+1)/2} + e_{(k+1)/2}\}$. This implies that $e_k = O(d)$, so the total imbalance after the recursion will be $O(d)$.

The explicit bound on e_k is somewhat difficult to characterize, but it is strictly greater than $4d - 1$ for sufficiently large k not equal to a power of two. For $k = 2^j$, the bound converges to $4d - 1$ from below. Hence, for most values of k , the bound obtained here is worse than the bound obtained in Theorem 1.

However, note that the bound arising from this proof will adapt to any theorems proved about balanced 2-colorings. For example, it was conjectured that there always exists an $O(\sqrt{d})$ -balanced 2-coloring [1,6]. If this were proved it would immediately imply the existence of an $O(\sqrt{d})$ -balanced k -coloring.

5 NP-Completeness Results

Akiyama and Urrutia [2] showed that every set of points on the 2-dimensional rectangular grid has an almost-balanced k -coloring for $k \geq 2$, and there is an efficient algorithm to compute such a coloring. They also gave an example of points on a 3-dimensional grid that do not admit an almost-balanced coloring. We strengthen this result by showing that testing whether a set of vertices has an almost-balanced 2-coloring is NP-complete for line families of dimension $d \geq 3$.

Theorem 3. *Let $d \geq 3$. Let $\mathcal{L}$ be a set of lines of dimension d containing a set of vertices P . Then the problem to decide whether P has an almost-balanced 2-coloring is NP-complete.*

Proof. Clearly, the problem is in NP, because given a 2-coloring, one can verify in polynomial time whether it is almost-balanced. We show the NP-hardness of the problem by reduction from NOT-ALL-EQUAL 3SAT which is known to be NP-hard [8]. The problem NOT-ALL-EQUAL 3SAT is the following: Given n Boolean variables $x_1, \dots, x_n$ and m clauses $c_1, \dots, c_m$ which each contain exactly three literals (i.e., variables or their negative), determine whether there exists an assignment of Boolean values to the variables such that for each clause at least one literal is true and at least one literal is false.

Given an instance S of NOT-ALL-EQUAL 3SAT, we want to construct a set of vertices P and a family $\mathcal{L}$ of lines containing the vertices P such that P can be almost-balanced 2-colored (with colors red and blue) if and only if S has a solution. For each clause c_j , we will have one line l_j that contains three vertices, one for each literal in c_j . The lines corresponding to different clauses use different vertices. In any almost-balanced coloring of P at least one of the vertices on any line must be red and at least one of the vertices must be blue.

Now, by adding additional lines and vertices, we will ensure that two vertices representing the same literal must have the same color, and two vertices representing a literal and its negation, respectively, must have different color. Two vertices p_1 and p_2 must have different color if we add a line containing exactly these two vertices. And they must have the same color if we add a line containing p_1 and a new vertex p_3 , and a line containing p_3 and p_2 .

This construction can be done in polynomial time. Note that each vertex is contained in at most three lines, i.e., $\mathcal{L}$ has dimension 3. $\square$

5.1 Geometric Settings

Now we want to strengthen the previous result for geometric settings where the vertices (or *points*) are placed on grid-lines in some dimension D , of which at most d intersect in one point. We suspect that finding an almost-balanced k -coloring is NP-complete in all geometric settings with $d \geq 3$, but leave this as an open problem.

Theorem 4. *The problem of finding an almost-balanced 2-coloring in a two-dimensional rectangular grid with one set of diagonals is NP-complete.*

Proof. (Sketch) We want to embed the construction in the proof of Theorem 3 into our grid. Figure 1 shows the construction for the formula $x_1x_2x_4 \vee x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_3\bar{x}_4$. The lines corresponding to a clause are vertical lines $6m + 1$ units apart (to avoid two points in two different columns lying on the same diagonal). Two literals of the same variable are not connected by a single line or two lines, but by a staircase-like arrangement of lines.

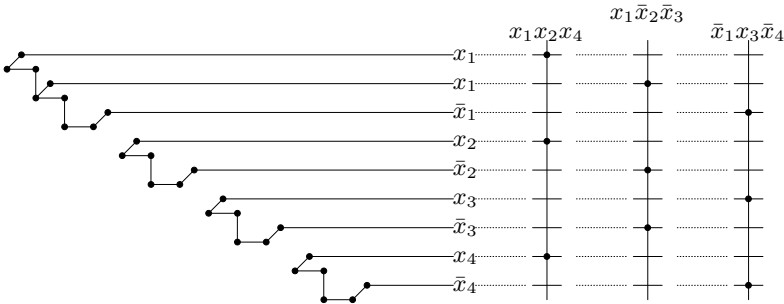


Fig. 1. Geometric construction for $x_1x_2x_4 \vee x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_3\bar{x}_4$. For readability the figure is not to scale

One immediately verifies that there exists an almost-balanced coloring for this construction if and only if the instance of NOT-ALL-EQUAL 3SAT has a solution. $\square$

Theorem 5. *The problem of finding an almost-balanced 2-coloring in a three-dimensional rectangular grid is NP-complete.*

Proof. (Sketch) The construction for 2-colorings of points in the three-dimensional rectangular grid is identical to the construction in the proof of Theorem 4, except that Fig. 1 should now be interpreted as a three-dimensional picture, with two planes depth. All points for clauses are added in the second (deeper) of these planes. $\square$

One interesting feature of this construction is that we “barely” use the third dimension, because we actually only use two parallel planes. Nevertheless, having two planes instead of one makes the problem NP-complete. Now we show that the problem does not become easier if we allow one more color.

Theorem 6. *The problem of finding an almost-balanced 3-coloring in a two-dimensional rectangular grid with one set of diagonals is NP-complete.*

Proof. We use almost the same reduction as in the proof of Theorem 4. The main difference is that having two points per grid line is not sufficient because this does not enforce any color. Therefore, for every grid line that contains two points in the above construction, we will add a third point. By adding even more points, we force these third points all to have the same color, say white. Hence, the two original points have exactly as much color choice as before, which means that the same reduction applies. For each column of a clause (which are the only grid lines containing three points) we add one point that also must be white. The remaining three points in such a column must all be red or blue (because their row contains now two more points, one of which is forced to be white), so as before, we must have at least one red and at least one blue point per column for a clause.

The precise addition of points works as follows. Assume that in the above construction, we have v grid lines that are vertical or diagonal and contain at least two points, and we have h horizontal grid lines that contain exactly two points.

Imagine placing v vertical diamonds, one attached to each other. See the left part in Fig. 2. Assume we have an almost-balanced 3-coloring of this construction. Let p be a tip of one diamond. The two points at the middle of this diamond both share a grid line with p , and because every grid line contains at most three points, they must have different colors from p . But then the other tip of the same color, which also shares grid lines with these points, must have the same color as p . Hence, all the tips of all diamonds have the same color.

Now add h horizontal diamonds, scaled such that they do not share any grid lines with the vertical diamonds, except at the attachment point. Again this construction has an almost-balanced 3-coloring, and all tips of all diamonds have the same color. See Fig. 2.

Assume that all tips of diamonds are colored white. In this construction, which we call a *color splitter*, there are then v rows and h columns that contain exactly one red and one blue point. These rows/columns are indicated with dashed lines in Fig. 2. Hence, if we place a third point in these rows/columns, then that point must be colored white.

All that remains to do is to place the color splitter in such a way that all these extra points can at the same time be in some grid line of the original construction. This can be done as follows. Start with the construction of the previous section. Extend all lines that contain at least two points infinitely. Place the color splitter such that it is below and to the right of any of the intersection points of these infinite lines. Now, for every horizontal infinite line

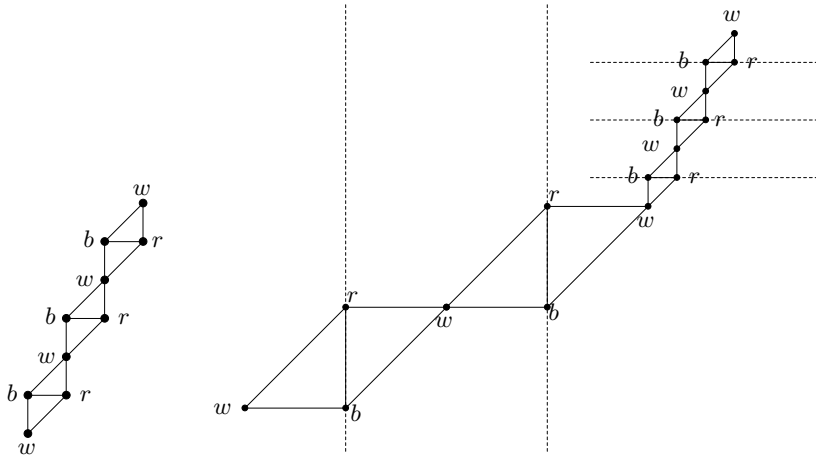


Fig. 2. The construction of a color splitter. We show here $v = 3$ and $h = 2$, though normally these numbers would be bigger

from the original construction, choose one of the h columns of the color splitter, and place a point at their intersection. Similarly, for any vertical infinite line or any diagonal infinite line of the original construction, choose one of the v rows of the color splitter and place a point at their intersection. See Fig. 3.

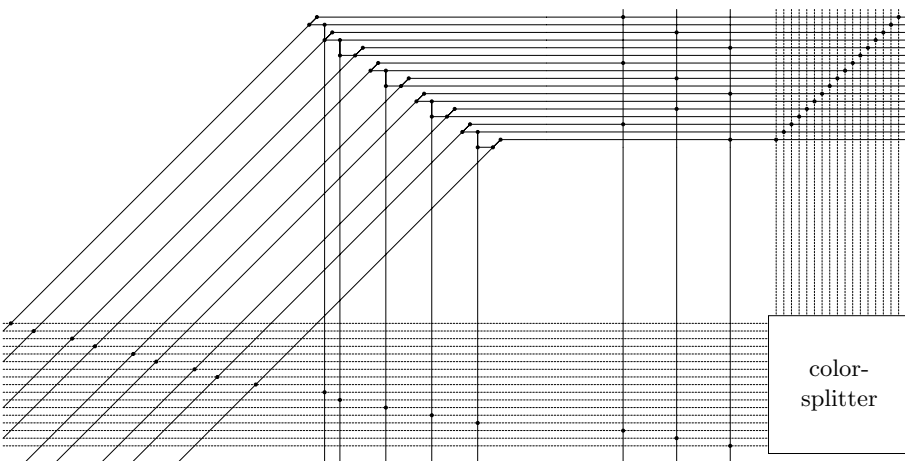


Fig. 3. Combining the construction for the 2-coloring with a color splitter

All these added points must be white. Hence, any of the grid lines of the original construction that contained two points before now must color these two points with red and blue. Hence, adding the third color does not give us any additional freedom, and the problem remains NP-complete. $\square$

We leave as an open problem whether finding an almost-balanced coloring is NP-hard for $k \geq 4$ colors on a rectangular grid with one set of diagonals. We would expect that the answer to this problem is yes, at least if we also increase d (the number of grid lines that are allowed to cross in one point). Observe that it would be enough to find a color splitter for $k \geq 4$ colors; if this exists, then the problem becomes NP-hard for k colors with a similar technique as in the previous section. We note however, that the construction of the color-splitter can be generalized to non-geometric settings with $d \geq \max\{3, k - 1\}$ so we have NP-completeness for these cases as well.

Acknowledgements. We thank Torben Hagerup for bringing Šíma's work [9] to our attention, and Joel Spencer for helpful comments.

References

1. Noga Alon and Joel H. Spencer. *The Probabilistic Method*. Wiley, New York, 1992. Chapter 12, pages 185–196.
2. Jin Akiyama and Jorge Urrutia. A note on balanced colourings for lattice points. *Discrete Mathematics*, 83(1):123–126, 1990.
3. Tetsuo Asano, Tomomi Matsui, and Takeshi Tokuyama. On the complexities of the optimal rounding problems of sequences and matrices. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT'00)*, Bergen, Norway, July 2000. To appear.
4. József Beck. Some results and problems in “combinatorial discrepancy theory”. In *Topics in Classical Number Theory: Proceedings of the International Conference on Number Theory*, pages 203–218, Budapest, Hungary, July 1981. Appeared in *Colloquia Mathematica Societatis János Bolyai*, volume 34, 1994.
5. József Beck and Tibor Fiala. “Integer-making” theorems. *Discrete Applied Mathematics*, 3:1–8, 1981.
6. József Beck and Vera T. Sós. Discrepancy theory. In *Handbook of Combinatorics*, volume 2, pages 1405–1446. Elsevier, Amsterdam, 1995.
7. A.J.W. Hilton and D. de Werra. A sufficient condition for equitable edge-colourings of simple graphs. *Discrete Mathematics*, 128(1-3): 179–201, 1994.
8. Thomas J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, San Diego, California, May 1978.
9. Jiří Šíma. Table rounding problem. *Comput. Artificial Intelligence*, 18(3): 175–189, 1999.
10. Joel Spencer. Geometric discrepancy theory. *Contemporary Mathematics*, 223, 1999.
11. Joel Spencer. Personal communication. 2000.

A Compositional Model for Confluent Dynamic Data-Flow Networks

Frank S. de Boer¹ and Marcello M. Bonsangue²

¹Utrecht University, The Netherlands
frankb@cs.uu.nl

²CWI, Amsterdam, The Netherlands
Marcello.Bonsangue@cw.nl

Abstract. We introduce a state-based language for programming dynamically changing networks which consist of processes that communicate asynchronously. For this language we introduce an operational semantics and a notion of observable which includes both partial correctness and absence of deadlock. Our main result is a compositional characterization of this notion of observable for a confluent sub-language.

1 Introduction

The goal of this paper is to develop a compositional semantics of a confluent subset of the language *MaC* (Mobile asynchronous Channels). *MaC* is an imperative programming language for describing the behavior of dynamic networks of asynchronously communicating processes.

A program in *MaC* consists of a (finite) number of generic process descriptions. Processes can be created dynamically and have an independent activity that proceeds in parallel with all the other processes in the system. They possess some internal data, which they store in *variables*. The value of a variable is either an element of a predefined data type or it is a *reference* to a *channel*. The variables of one process are not accessible to other processes. The processes can interact only by sending and receiving messages *asynchronously* via channels which are (unbounded) FIFO buffers. A message contains exactly one value; this can be a value of some given data type, like integer or a boolean, or it can be a reference to a channel. Channels are created dynamically. In fact, the creation of a process consists of the creation of a channel which connects it with its creator. This channel has a unique identity which is initially known only to the created process and its creator. As with any channel, the identity of this initial channel too can be communicated to other processes via other channels. Thus we see that a system described by a program in the language *MaC* consists of a dynamically evolving network of processes, which are all executing in parallel, and which communicate asynchronously via mobile channels. In particular, this means that the communication structure of the processes, i.e. which processes are connected by which channels, is completely dynamic, without any regular structure imposed on it a priori.

For *MaC* we first introduce a simple operational semantics and the following notion of observable. Let Σ denote the set of (global) states. A global state specifies, for each existing process, the values of its variables, and, for each existing channel, the contents of its buffer. The semantics $\mathcal{O}$ assigns to each program ρ in *MaC* a *partial* function in $\Sigma \rightarrow \mathcal{P}(\Sigma)$ such that $\mathcal{O}(\rho)(\sigma)$ collects all final results of successfully terminating computations in σ , if ρ does not have a deadlocking computation starting from σ . Otherwise, $\mathcal{O}(\rho)(\sigma)$ is undefined.

This notion of observable $\mathcal{O}$ provides a semantic basis for the following interpretation of a correctness formula $\{\phi\}\rho\{\psi\}$ in Hoare logic: every execution of program ρ in a state which satisfies the assertion ϕ does *not deadlock* and upon termination the assertion ψ will hold. An axiomatization of this interpretation of correctness formulas thus requires a method for proving absence of deadlock.

In this paper we identify a *confluent* sub-language of *MaC* which allows to abstract from the order between the communications of different processes and the order between the communications on different channels within a process [14, 15]. A necessary condition for obtaining a confluent sub-language is the restriction to local non-determinism and to channels which are uni-directional and one-to-one. In a dynamic network of processes the restriction to such channels implies that at any moment during the execution of a program for each existing channel there are at most two processes whose internal data contain a reference to it; one of these processes only may use this reference for sending values and the other may use this reference only for receiving values.

For confluent *MaC* programs we develop a compositional characterization of the semantics $\mathcal{O}$. It is based on the local semantics of each single process, which includes information about the channels it has created and, for each known channel, information about the sequence of values the process has sent or read. Information about the deadlock behavior of a process is given in terms of a singleton ready set including a channel reference. As such we do not have any information about the order between the communications of a process on different channels and the order between the communications of different processes. In general, this abstraction will in practice simplify reasoning about the correctness of distributed systems.

Comparison with related work: The language *MaC* is a sub-language of the one introduced in [3]. The latter is an abstract core for the Manifold coordination language [4]. The main feature relevant in this context is anonymous communication, in contrast with parallel object-oriented languages and actor languages, as studied, for example, in [5] and [1], where communication between the processes, i.e., objects or actors, is established via their identities.

In contrast to the π -calculus [16] which constitutes a process algebra for mobility, our language *MaC* provides a state-based model for mobility. As such our language provides a framework for the study of the semantic basis of assertional proof methods for mobility. *MaC* can also be seen as a dynamic version of asynchronous CSP [14]. In fact, the language *MaC* is similar to the verification modeling language Promela [12], a tool for analyzing the logical consistency of distributed systems, specifically of data communication protocols. However, the

semantic investigations of Promela are performed within the context of temporal logic, whereas *MaC* provides a semantic basis for Hoare logics.

Our main result can also be viewed as a generalization of the compositional semantics of Kahn (data-flow) networks [13] (where the number of processes and the communication structure is fixed). Instead of a function the communication behavior of a process in the language *MaC* is specified in terms of a relation between the sequence of values it inputs and the sequence of values it outputs. This information suffices because of the restriction to confluent programs. Confluence has been studied also in the context of concurrent constraint programming [9] where mobility is modeled in terms of logical variables.

Generalization of Kahn (data-flow) networks for describing dynamically reconfigurable or mobile networks have also been studied in [6] and [11] using the model of stream functions. In this paper we study a different notion of observable which includes partial correctness and absence of deadlock. Furthermore, our language includes both dynamic process and channel creation. On the other hand, we restrict to confluent dynamic networks.

2 Syntax and Operational Semantics

A program in the language *MaC* is a (finite) collection of generic process descriptions. Such a generic process description consists of an association of a unique name P , the so-called *process type*, with a statement describing generically the behavior of its instances.

The statement associated with a process type P is executed by a process, i.e. an instance of that process type. Such a process acts upon some internal data that are stored in *variables*. The variables of a process are *private*, i.e., the data stored in the variables of a process is not accessible by another process, even if both processes are of the same type. We denote by *Var*, with its typical elements $x, y, \dots$, the set of variables. The value of a variable can be either an element of a predefined data type, like integer or boolean, or a reference to a channel.

We have the following repertoire of basic actions of a process:

$$x := e \qquad x := \text{new}(P) \qquad x!y \qquad x?y$$

The execution of an assignment $x := e$ by a process consists of assigning the value resulting from evaluation of the expression e to the variable x (we abstract here from the internal structure of e and assume that its evaluation is deterministic and always terminates).

The execution of the statement $x := \text{new}(P)$ by a process consists of the creation of a new process of type P and a new channel which, initially, forms a link between the two (creator and created) processes. A reference to this channel will be stored in the variable x of the creator and to a distinguished variable *chn* of the created process. The newly created process starts executing the statement associated with P in parallel with all the other existing processes.

Processes can interact only by sending and receiving messages via channels. A message contains exactly one value; this can be of any type, including channel

references. We restrict in this paper to asynchronous channels that are implemented by (unbounded) FIFO buffers. The execution of the output action $x!y$ sends the value stored in the variable y to the channel referred to by the variable x . The execution of the input action $x?y$ suspends until a value is available through the specified channel. The value read is removed from the channel and then stored in the variable y .

The set of statements, with typical element S , is generated by composing the above basic actions using well-known sequential non-deterministic programming constructs [8]. A program ρ is a finite collection of generic process descriptions of the form $P \Leftarrow S$. The execution of a program $\{P_0 \Leftarrow S_0, \dots, P_n \Leftarrow S_n\}$ starts with the execution of a root-process of type P_0 .

Next we define formally the (operational) semantics of the programming language by means of a transition system. We assume given an infinite set C of channel identities, with typical elements $c, c', \dots$. The set Val , with typical elements $u, v, \dots$, includes the set C of channel identities and the value $\perp$ which indicates that a variable is ‘uninitialized’.

A *global state* σ of a network of processes specifies the existing channels, that is, the channels that have already been created, and the contents of their buffers. Formally, σ is a partial function in $C \rightarrow Val^*$ (here Val^* denotes the set of finite sequences of elements in Val). Its domain $dom(\sigma) \subseteq C$ is a finite set of channel identities, representing those channels which have been created. Moreover, for every existing channel $c \in dom(\sigma)$, the contents of its buffer is specified by $\sigma(c) \in Val^*$. On the other hand, the internal state $s \in Var \rightarrow Val$ of a process simply specifies the values of its variables.

The behavior of a network of processes is described in terms of a transition relation between configurations of the form $\langle X, \sigma \rangle$, where σ is the global state of the existing channels and X is a finite multiset of pairs of the form (S, s) , for some internal state s and statement S . A pair of the form (S, s) denotes an active process within the network: its current internal state is given by s and S denotes the statement to be executed. We have the following transitions for the basic actions (we assume given a program ρ). Below the operation of multiset union is denoted by $\uplus$ and by nil we denote the empty statement.

Assignment: $\langle X \uplus \{(x := e, s)\}, \sigma \rangle \rightarrow \langle X \uplus \{(nil, s[s(e)/x])\}, \sigma \rangle$, where $s(e)$ denotes the value of e in s and $s[v/x]$ denotes the function mapping x to v and otherwise acting as s .

Creation: $\langle X \uplus \{(x := \text{new}(P), s)\}, \sigma \rangle \rightarrow \langle X \uplus \{(nil, s[c/x]), (S, s_0[c/chn])\}, \sigma' \rangle$, where $P \Leftarrow S$ occurs in ρ and $\sigma' = \sigma[\varepsilon/c]$ for some $c \in C \setminus dom(\sigma)$. Thus σ' extends σ by mapping the new channel c to the empty sequence ε . Moreover, the initial state of the newly created process s_0 satisfies the following: $s_0(x) = \perp$, for every variable $x \in Var$. Note that the new channel c forms a link between the two processes. The statement S is the one associated with the process type P in the program ρ .

Input: Let $s(x) = c(\neq \perp)$. $\langle X \uplus \{(x?y, s)\}, \sigma \rangle \rightarrow \langle X \uplus \{(nil, s[u/y])\}, \sigma' \rangle$, where $\sigma(c) = w \cdot u$ for some $u \in Val$, and σ' results from σ by removing u from the buffer of c , that is, $\sigma' = \sigma[w/c]$.

Output: Let $s(x) = c(\neq \perp)$ in $\langle X \uplus \{(x!y, s)\}, \sigma \rangle \rightarrow \langle X \uplus \{(\text{nil}, s)\}, \sigma' \rangle$, where σ' results from σ by adding the value $s(y)$ to the sequence $\sigma(c)$, that is, $\sigma' = \sigma[s(y) \cdot \sigma(c)/c]$.

The remaining transition rules for compound statements are standard and therefore omitted. By $\rightarrow^*$ we denote the reflexive transitive closure of $\rightarrow$ and $\langle X, \sigma \rangle \Rightarrow \delta$ indicates the existence of a deadlocking computation starting from $\langle X, \sigma \rangle$, that is, $\langle X, \sigma \rangle \rightarrow^* \langle X', \sigma' \rangle$ with X' containing at least one pair (S, s) such that $S \neq \text{nil}$, and from the configuration $\langle X', \sigma' \rangle$ no further transition is possible. Moreover, $\langle X, \sigma \rangle \Rightarrow \langle X', \sigma' \rangle$ indicates a successfully terminating computation with final configuration $\langle X', \sigma' \rangle$, that is, $\langle X, \sigma \rangle \rightarrow^* \langle X', \sigma' \rangle$ and X' contains only pairs of the form (nil, s) .

We are now in a position to introduce the following notion of observable.

Definition 1. Let $\rho = \{P_0 \Leftarrow S_0, \dots, P_n \Leftarrow S_n\}$ be a program. By $\langle X_0, \sigma_0 \rangle$ we denote its initial configuration $\langle \{(S_0, s_0)\}, \sigma_0 \rangle$, where $s_0(x) = \perp$, for every variable x , and $\text{dom}(\sigma_0) = \emptyset$. We define

$$\mathcal{O}(\rho) = \begin{cases} \delta & \text{if } \langle X_0, \sigma_0 \rangle \Rightarrow \delta \\ \{ \langle X, \sigma \rangle \mid \langle X_0, \sigma_0 \rangle \Rightarrow \langle X, \sigma \rangle \} & \text{otherwise} \end{cases}$$

Note that thus $\mathcal{O}(\rho) = \delta$ indicates that ρ has a deadlocking computation. On the other hand, if ρ does not have a deadlocking computation then $\mathcal{O}(\rho)$ collects all the final configurations of successfully terminating computations.

3 Compositionality

In this section we introduce, for a certain kind of programs, a compositional characterization of the notion of observable defined in the previous section.

First of all we restrict to local non-determinism. Moreover, we assume now a typing of the variables: we have variables of some predefined data types and we assume channel variables to be either of type ι , for input, and o , for output. Let $\bar{C}$, with typical element $\bar{c}$, be a copy of C . A channel variable of type ι always refers to an element of C , whereas, a channel variable of type o always refers to an element of $\bar{C}$. (The set of all possible values thus includes both C and $\bar{C}$.) We restrict to programs which are well-typed. In particular, in an output $x!y$ the variable x is of type o and in an input $x?y$ the variable x is of type ι . An input $x?y$ now also suspends if the value to be read is not of the same type as the variable y . Moreover, we assume that the distinguished variable chn (used for storing the initial link with the creator) is of type o . Consequently, in $x := \text{new}(P)$ the variable x has to be of type ι . In other words, initially, the flow of information along the newly created channel goes from the created to the creator process.

Finally, we assume that an output $x!y$, where y is a channel variable, is immediately followed by an assignment which uninitializes the variable y , i.e. it sets y to $\perp$. But for this latter, we do not allow channel variables (either of type ι or o) to appear in an assignment. As a result, channels are one-to-one and uni-directional.

We extend now the notion of an internal state s to include the following information about the channels. Let $\gamma \notin \text{Val}$ and $\text{Val}_\gamma = \text{Val} \cup \{\gamma\}$. For each channel $c \in C$, $s(c) \in \text{Val}_\gamma^*$ denotes, among others, the sequence of values received from channel c , and $s(\bar{c}) \in \text{Val}_\gamma^*$, denotes, among others, the sequence of values sent along channel c . More precisely, in a sequence $w_1 \cdot \gamma \cdot w_2 \cdot \gamma \cdots$, the symbol γ indicates that first the sequence of values w_1 has been sent along c (or received from c) and that after control over this channel has been released and subsequently regained again the sequence w_2 has been sent (or received), etc.. Note that a process releases control over a channel only when it outputs that channel and that it subsequently may again regain control over it only by receiving it via some input.

Additionally, we introduce a component $s(\nu) \in (C \cup \{\perp\}) \times \mathcal{P}(C)$. The first element of $s(\nu)$ indicates the channel which initially links the process with its creator (in case of the root-process we have here $\perp$). The second element of $s(\nu)$ indicates the set of channels which have been created by the process itself.

Given this extended notion of an internal state of a process we now present the transitions describing the execution of the basic actions with respect to the internal state of a process (we omit the standard transition for a simple assignment).

Creation: Let $s(\nu) = (u, V)$ and $c \notin V$ in $\langle x := \text{new}(P), s \rangle \rightarrow \langle \text{nil}, s'[c/x] \rangle$. Here s' results from s by adding c , that is, $s'(\nu) = (u, V \cup \{c\})$. The only effect at the local level of the execution of a basic action $x := \text{new}(P)$ is the assignment to x of a channel c which is new with respect to the set of channels already created by the process.

Output 1: If $s(x) = \bar{c}$ and y is not a channel variable, i.e. y is of some given data type like the integers or booleans, then $\langle x!y, s \rangle \rightarrow \langle \text{nil}, s[s(\bar{c}) \cdot s(y)/\bar{c}] \rangle$. The local effect of an output (of a value of some predefined data type) consists of adding the value stored in the variable y to the sequence of values already sent.

Output 2: If $s(x) = \bar{c}$ then $\langle x!y, s \rangle \rightarrow \langle \text{nil}, s[s(\bar{c}) \cdot v/\bar{c}][s(v) \cdot \gamma/v] \rangle$, where $v = s(y)$ and y is a channel variable. So after the output along the channel c of the value v stored in the variable y , first the value v is appended to $s(\bar{c})$, which basically records the sequence of values sent along the channel $\bar{c}$. Finally, the output of channel v (and consequently its release) is recorded as such by γ in the sequence $s(v)$ which records the sequence of values sent along the channel v , in case $v \in \bar{C}$, and received from it, in case $v \in C$. Note that we have to perform the state-changes indicated by $[s(\bar{c}) \cdot v/\bar{c}]$ and $[s(v) \cdot \gamma/v]$ in this order to describe correctly the case that $v = \bar{c}$.

Input: If $s(x) = c(\neq \perp)$ then $\langle x?y, s \rangle \rightarrow \langle \text{nil}, s[v/y, s(c) \cdot v/c] \rangle$, where $v \in \text{Val}$ is an *arbitrary* value (of the same type as y). This value is assigned to y and appended to the sequence $s(c)$ of values received so far (along channel c). Note that because channels are one-to-one and unidirectional it cannot be the case that $v = c$.

On the basis of the above transition system (we omit the rules from compound statement since they are standard) we define the operational semantics of statements as follows.

Definition 2. An (extended) initial state s satisfies the following: for some $u \in C \cup \{\perp\}$ we have that $s(\text{chn}) = u$, and $s(x) = \perp$, for every other variable, moreover, $s(d) = s(\bar{d}) = \epsilon$, for every channel d , and, finally, $s(\nu) = (u, \emptyset)$. We define $\mathcal{O}(S) = \langle T, R \rangle$, where $T = \{s' \mid \langle S, s \rangle \rightarrow^* \langle \text{nil}, s' \rangle \text{ for some initial state } s\}$ and $R = \{(s', s(x), t(y)) \mid \langle S, s \rangle \rightarrow^* \langle x?y; S', s' \rangle, \text{ for some initial state } s\}$ (here $t(y)$ denotes the type of y).

The component T in the semantics $\mathcal{O}(S)$ collects all the final states of successfully terminating (local) computations of S (starting from an initial state). The component R , on the other hand, collects all the intermediate states where control is about to perform an input, plus information about the channel involved and the type of the value to be read. The restriction to local non-determinism implies that when an input $x?y$ is about to be executed, it will always appear in a context of the form $x?y; S$ for some (possibly empty) statement S (no other inputs are offered as an alternative).

The information in R corresponds with the well-known concept of the *ready sets* [17] and will be used for determining whether a program (containing a process type $P \Leftarrow S$) has a deadlocking computation.

Our compositional semantics is based on the *compatibility* of a set of internal states (without loss of generality we may indeed restrict to sets rather than multisets of *extended* internal states s because of the additional information $s(\nu)$). In order to define this notion we use the set $C_\perp = C \cup \{\perp\}$, ranged over by $\alpha, \beta, \dots$, to identify processes. The idea is that the channel which initially links the created process with its creator will be used to identify the created process itself ($\perp$ will be used to identify the root-process). We use these process identifiers in finite sequences of labeled inputs $(\alpha, c?v)$ and outputs $(\alpha, c!v)$ to indicate the process involved in the communication. Given such a sequence h and a channel $c \in C$ we denote by $\text{sent}(h, c)$ the sequence of values in Val sent to the channel c and by $\text{rec}(h, c)$ the sequence of values in Val received from the channel c .

A *history* h is a (finite) sequence of labeled inputs $(\alpha, c?v)$ and outputs $(\alpha, c!v)$ which satisfies the following.

Prefix invariance: For every prefix h' of h and channel c we have that the sequence $\text{rec}(h', c)$ of values delivered by c is a prefix of the sequence $\text{sent}(h', c)$ of values received by channel c .

Input ownership: For every prefix $h_0 \cdot (\alpha, c?v)$ of h , either the process α owns the input of the channel c in h_0 , or $h_0 = h_1 \cdot (\alpha, d?c) \cdot h_2$ for some channel d distinct from c and α owns the input of the channel c in h_2 . A process α is said to be the owner of the input of a channel c in a sequence h if, for any channel e , there is no occurrence in h of an output $(\alpha, e!c)$, and for every occurrence in h of an input $(\beta, c?w)$ we have $\alpha = \beta$.

Output ownership: For every prefix $h_0 \cdot (\alpha, c!v)$ of h , either the process α owns the output of the channel c in h_0 , or $h_0 = h_1 \cdot (\alpha, d?c) \cdot h_2$ for some channel d (not necessarily distinct from c) and α owns the output of the channel c in h_2 . A process α is said to be the owner of the output of a channel c in a sequence

h if for any channel e there is no occurrence in h of an output $(\alpha, e!c)$, and for every occurrence in h of an output $(\beta, c!w)$ we have $\alpha = \beta$.

Input/output ownership essentially states that a process can communicate along a channel only if either it is the first user of that channel or it has received that channel via a preceding communication. Moreover, exclusive control over a channel is released only when that channel is outputted.

We can obtain the local information of a process from a given history as follows. For a history h , an internal state s , we write $s \simeq h$ if $s(\nu) = (\alpha, V)$ implies, for every channel c , both $s(c) = in(h, \alpha, c)$ and $s(\bar{c}) = out(h, \alpha, c)$, where $in(h, \alpha, c)$ and $out(h, \alpha, c)$ denote the sequences of values received from and sent to the channel c by the process α as recorded by the history h . Occurrences of γ in those sequences will denote release of control of the channel c by the process α . More specifically, we have $in((\alpha, d!c) \cdot h, \alpha, c) = \gamma \cdot in(h, \alpha, c)$ and similarly for $out((\alpha, d!c) \cdot h, \alpha, c)$. Thus $s \simeq h$ basically states that the information about the communication behavior in the internal state s is compatible with the information given by the history h . The *compatibility* of h with respect to a set of internal states X is defined below.

Definition 3. Let h be a history and X be a finite set of internal states. We say that h is compatible with X if the following two conditions hold:

1. for every $s \in X$, $s \simeq h$;
2. there exists a finite tree (the tree of creation) with X as nodes such that
 - if s is the root of the tree then $s(\nu) = (\perp, V)$, for some $V \subseteq C$;
 - if $s \in X$ with $s(\nu) = (u, V)$ then for all $v \in V$ there exists a unique direct descendent node $s' \in X$ with $s'(\nu) = (v, W)$, for some $W \subseteq C$.

The existence of a tree of creation ensures the uniqueness of the name of the created channels. It is worthwhile to observe that it is not sufficient to require disjointness of the names used by any two distinct existing processes, as this does not exclude cycles in the creation ordering (for example, two processes creating each other).

Let h be a history compatible with a finite set of (local) states X . For each channel c which appears in X , we denote by $own(h, c)$ and $own(h, \bar{c})$ the sequences of processes who had the ownership of the reference for inputting from and outputting to the channel c , respectively.

For a given set of (local) states X there may be several histories, each of them compatible with X . The next theorem specifies the relevant information recorded in a history.

Theorem 1. Let X be a finite set of (local) states, and h_1 and h_2 be two histories compatible with X . For all process id's α and channels c the following holds:

1. $in(h_1, \alpha, c) = in(h_2, \alpha, c)$ and $out(h_1, \alpha, c) = out(h_2, \alpha, c)$;
2. $sent(h_1, c) = sent(h_2, c)$ and $rec(h_1, c) = rec(h_2, c)$;
3. $own(h_1, c) = own(h_2, c)$ and $own(h_1, \bar{c}) = own(h_2, \bar{c})$.

This theorem states that the compatibility relation abstracts from the order of communication between different channels in a global history. For example, even the ordering between inputs and outputs on different channels is irrelevant. This contrasts with the usual models of asynchronous communicating non-deterministic processes [14,15]. This abstraction is made possible because of the restriction to confluent programs.

In order to formulate the main theorem of this paper we still need some more definitions. We say that a set X of extended internal states is *consistent* if there exists a history h compatible with X . Given a consistent set X of extended internal states s , we denote by $\text{conf}(X)$, the corresponding (final) configuration $\langle \tilde{X}, \sigma \rangle$. That is, $\tilde{X}$ consists of those pairs $(\text{nil}, \tilde{s})$ for which there exists $s \in X$ such that $\tilde{s}$ is obtained from s by removing the additional information about the communicated values and the created channels. The global state σ derives from a history h compatible with X in the obvious way (i.e. by mapping every channel c such that $s(\nu) = (c, V)$ for some $s \in X$ and $V \subseteq C$, to the sequence obtained by deleting the prefix $\text{rec}(h, c)$ from $\text{sent}(h, c)$). Note that the above Theorem 1 guarantees that σ is indeed well-defined.

Definition 4. We assume given T_i and R_i , for $i = 1, \dots, n$, with T_i a set of (extended) internal states and R_i a set of triples of the form (s, c, t) , where s is an extended internal state, c is a channel and t is a type (of the value to be read from c in the state s).

We denote by $\bigsqcup_i T_i$ the set of final configurations $\text{conf}(X)$ such that the set X of (extended) internal states is consistent and every state s in X belongs to some T_i . Additionally, for some state $s \in T_0$ we have $s(\nu) = (\perp, V)$, for some $V \subseteq C$.

Analogously, by $\bigsqcup_i \langle T_i, R_i \rangle$ we denote the set of final configurations $\text{conf}(X)$ such that X is consistent, and there exists a state s in X that does not belong to any T_i , and, finally, every state s in X either belongs to some T_i or there exists a triple $(s, c, t) \in R_i$ such that either $\sigma(c) = \epsilon$ or the first value of $\sigma(c)$ is not of type t .

Abstracting from the control information, the set of configurations $\bigsqcup_i \langle T_i, R_i \rangle$ in fact describes all possible deadlock configurations, whereas $\bigsqcup_i T_i$ describes all the final configurations of successfully terminating computations of the given program. Finally, we are in a position to formulate the main theorem of this paper.

Theorem 2. Let $\rho = \{P_0 \Leftarrow S_0, \dots, P_n \Leftarrow S_n\}$ and $\mathcal{O}(S_i) = \langle T_i, R_i \rangle$, $i = 0, \dots, n$. We have that

$$\mathcal{O}(\rho) = \begin{cases} \bigsqcup_i T_i & \text{if } \bigsqcup_i \langle T_i, R_i \rangle = \emptyset \\ \delta & \text{otherwise.} \end{cases}$$

Thus the observable behavior of a confluent *MaC* program can be obtained in a compositional manner from the local semantics of the statements of each process description of the program. The information of the ready sets of each local semantics is used to determine if the program deadlocks.

4 Conclusion and Future Work

To the best of the authors knowledge, we have presented a first state-based semantics for a confluent language for mobile data-flow networks which is compositional with respect to the abstract notion of observable considered in this paper. This notion of observable is more abstract than the bisimulation-based semantics for most action-based calculi for mobility [16,10,7], and the trace-based semantics for state-based languages [12].

The proposed semantics will be used for defining a compositional Hoare logic for confluent *MaC* programs along the lines of [5]. The fact that the order between the communications between different processes and the communication on different channels within a process is semantically irrelevant will in general simplify the correctness proofs.

References

1. G. Agha, I. Mason, S. Smith, and C. Talcott. A foundation for actor computation *Journal of Functional Programming*, 1(1):1-69, 1993.
2. R. Amadio, I. Castellani, and D. Sangiorgi. On Bisimulations for the Asynchronous π -calculus. *Theoretical Computer Science*, 195:291-324, 1998.
3. F. Arbab, F.S. de Boer, and M.M. Bonsangue. A coordination language for mobile components. In *Proc. of SAC 2000*, pp. 166-173, ACM press, 2000.
4. F. Arbab, I. Herman, and P. Spilling. An overview of Manifold and its implementation. *Concurrency: Practice and Experience*, 5(1):23-70, 1993.
5. F.S. de Boer. Reasoning about asynchronous communication in dynamically evolving object structures. To appear in *Theoretical Computer Science*, 2000.
6. M. Broy. Equations for describing dynamic nets of communicating systems. In *Proc. 5th COMPASS workshop*, vol. 906 of LNCS, pp. 170-187, 1995.
7. L. Cardelli and A.D. Gordon. Mobile ambients. In *Proc. of Foundation of Software Science and Computational Structures*, vol. 1378 of LNCS, pp. 140-155, 1998.
8. E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
9. M. Falaschi, M. Gabbriellini, K. Marriot, and C. Palamidessi. Confluence in concurrent constraint programming. In *Theoretical Computer Science*, 183(2), 1997.
10. C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join calculus. In *Proc. POPL'96*, pp. 372-385, 1996.
11. R. Grosu and K. Stølen. A model for mobile point-to-point data-flow networks without channel sharing. In *Proc. AMAST'96*, LNCS, 1996.
12. G.J. Holzmann. The model checker SPIN *IEEE Transactions on Software Engineering* 23:5, 1997.
13. G. Kahn. The semantics of a simple language for parallel programming. In IFIP74 Congress, North Holland, Amsterdam, 1974.
14. He Jifeng, M.B. Josephs, and C.A.R. Hoare. A theory of synchrony and asynchrony. In *Proc. IFIP Conf. on Programming Concepts and Methods*, 1990.
15. B. Jonsson. A fully abstract trace model for dataflow and asynchronous networks. *Distributed Computing*, 7:197-212, 1994.
16. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation* 100(1):1-77, 1992.
17. E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica* 23:9-66, 1986.

Restricted Nondeterministic Read-Once Branching Programs and an Exponential Lower Bound for Integer Multiplication (Extended Abstract)

Beate Bollig*

FB Informatik, LS2, Univ. Dortmund,
44221 Dortmund, Germany
bollig@ls2.cs.uni-dortmund.de

Abstract. Branching programs are a well established computation model for Boolean functions, especially read-once branching programs have been studied intensively. In this paper the expressive power of nondeterministic read-once branching programs, i.e., the class of functions representable in polynomial size, is investigated. For that reason two restricted models of nondeterministic read-once branching programs are defined and a lower bound method is presented. Furthermore, the first exponential lower bound for integer multiplication on the size of a nondeterministic nonoblivious read-once branching program model is proven.

1 Introduction and Results

Branching programs (BPs) or Binary Decision Diagrams (BDDs) are a well established representation type or computation model for Boolean functions.

Definition 1. *A branching program (BP) or binary decision diagram (BDD) on the variable set $X_n = \{x_1, \dots, x_n\}$ is a directed acyclic graph with one source and two sinks labelled by the constants 0 or 1, resp. Each non-sink node (or inner node) is labelled by a Boolean variable and has two outgoing edges one labelled by 0 and the other by 1. At each node v a Boolean function $f_v : \{0, 1\}^n \rightarrow \{0, 1\}$ is represented. A c -sink represents the constant function c . If f_{v_0} and f_{v_1} are the functions at the 0- or 1-successor of v , resp., and v is labelled by x_i , f_v is defined by Shannon's decomposition rule $f_v(a) := \bar{a}_i f_{v_0}(a) \vee a_i f_{v_1}(a)$. The computation path for the input a in a BP G is the sequence of nodes visited during the evaluation of a in G .*

The size of a branching program G is equal to the number of its nodes and is denoted by $|G|$. $\text{BP}(f)$ denotes the size of the smallest BP for a function f . The depth of a branching program is the maximum length of a path from the source to one of the sinks.

* Supported in part by DFG grant We 1066/9.

The branching program size of Boolean functions f is known to be a measure for the space complexity of nonuniform Turing machines and known to lie between the circuit size of f and its $\{\wedge, \vee, \neg\}$ -formula size (see, e.g., [21]). 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 develop and strengthen lower bound techniques one considers restricted computation models.

Definition 2. *i) A branching program is called read k times (BP k) if each variable is tested on each path at most k times.*
ii) A BP is called oblivious if the node set can be partitioned into levels such that edges lead from lower to higher levels and all inner nodes of one level are labelled by the same variable.

Read-once branching programs (BP1s) have been investigated intensively. Borodin, Razborov, and Smolensky [6] have proved one of the first exponential lower bounds for BP k s. For oblivious branching programs of restricted depth exponential lower bounds have been proved, e.g., by Alon and Maass [2]. Nondeterminism is one of the most powerful concepts in computer science. In analogy to the definition for Turing machines, different modes of acceptance can be studied for branching programs. The definition of Ω -branching programs due to Meinel [15] summarizes the most interesting modes of acceptance.

Definition 3. *Let Ω be a set of binary Boolean operations. An Ω -branching program on the variable set $X_n = \{x_1, \dots, x_n\}$ is a directed acyclic graph with decision nodes for Boolean variables and nondeterministic nodes. Each nondeterministic node is labelled by some function $\omega \in \Omega$ and has two outgoing edges labelled by 0 and 1, resp. A c -sink represents the constant c . Shannon's decomposition rule is applied at decision nodes. If f_{v_0} and f_{v_1} are the functions at the 0- or 1-successor of v , resp., and v is labelled by ω , the function $f_v = \omega(f_{v_0}, f_{v_1})$ is represented at v .*

Definitions of nondeterministic variants of restricted BPs are derived in a straightforward way by requiring that the decision nodes fulfill the usual restrictions as for deterministic BPs. In the following if nothing else is mentioned nondeterministic BPs means OR-BPs. The results of Borodin, Razborov, and Smolensky [6] for BP k s hold (and have been stated by the authors) also for OR-BP k s. Moreover, Thathachar [20] has proved an exponential gap between the size of OR-BP k s and deterministic BP $(k + 1)$ s. Besides this complexity theoretical viewpoint people have used branching programs in applications. In his seminal paper Bryant [7] introduced ordered binary decision diagrams (OBDDs) which are up to now the most popular representation for formal circuit verification.

Definition 4. *Let $X_n = \{x_1, \dots, x_n\}$ be a set of Boolean variables. A variable ordering π on X_n is a permutation on $\{1, \dots, n\}$ leading to the ordered list $x_{\pi(1)}, \dots, x_{\pi(n)}$ of the variables.*

- i) A π -OBDD for a variable ordering π is a BP where the sequence of tests on a path is restricted by the variable ordering π , i.e., if an edge leads from an x_i -node to an x_j -node, the condition $\pi(i) < \pi(j)$ has to be fulfilled.
- ii) An OBDD is a π -OBDD for some variable ordering π .

Unfortunately, several important and also quite simple functions have exponential OBDD size. Therefore, more general representations with good algorithmic behavior are necessary. Gergov and Meinel [10] and Sieling and Wegener [19] have shown independently how read-once branching programs can be used for verification. In order to obtain efficient algorithms for many operations they define a more general variable ordering.

Definition 5. A graph ordering is a branching program with a single sink labelled end. On each path from the source to the sink there is for each variable x_i exactly one node labelled x_i . A graph ordering G_0 is called a tree ordering if G_0 becomes a tree of polynomial size by eliminating the sink and replacing multiedges between nodes by simple edges.

A graph-driven (tree-driven) BP1 with respect to a graph ordering G_0 (tree ordering T_0), G_0 -BP1 (T_0 -BP1) for short, is a BP1 with the following additional property. For an arbitrary input $a \in \{0, 1\}^n$, let $\mathcal{L}(a)$ be the list of labels at the nodes on the computation path for a in the BP1 and similarly let $\mathcal{L}_0(a)$ be the list of labels on the computation path for a in G_0 (T_0). We require that $\mathcal{L}(a)$ is a subsequence of $\mathcal{L}_0(a)$.

It is easy to see that an arbitrary read-once branching program G is ordered with respect to a suitable chosen graph ordering. Sieling and Wegener [19] have shown that sometimes tree-driven BP1s have nicer algorithmic properties.

Nondeterministic concepts also may be useful for applications. Partitioned BDDs (PBDDs) introduced by Jain, Bitner, Fussell, and Abraham [13] are obtained by imposing strong structural restrictions on nondeterministic read-once branching programs.

Definition 6. A k -PBDD (partitioned BDD with k parts where k may depend on the number of variables) consists of k OBDDs whose variable orderings may be different. The output value for an input a is defined as 1 iff at least one of the OBDDs computes 1 on a . A PBDD is a k -PBDD for some k . The size of a k -PBDD is the sum of the sizes of the k OBDDs.

Now, we present a new restricted nondeterministic read-once branching program model which allows us to bound the power of nondeterminism.

Definition 7. A nondeterministic graph-driven BP1 (tree-driven BP1), OR- G_0 -BP1 (OR- T_0 -BP1) for short, is a nondeterministic BP1 where the Boolean variables labelling the decision nodes are ordered according to a graph ordering (tree ordering).

In the rest of this section we motivate our results. A nondeterministic Turing machine can be simulated in polynomial time by a so-called *guess-and-verify*

machine. It is an open question whether the analogous simulation exists in the context of space-bounded computation. Sauerhoff [17] has given a negative answer to this question for OR-OBDDs. The requirement that all nondeterministic nodes are at the top of the OBDD may blow-up the size exponentially. The question is still open for OR-BP1s and seems to be difficult to answer. The known lower bound techniques for OR-BP1s are not subtle enough to prove exponential lower bounds on the size of *guess-and-verify* BP1s for functions representable by OR-BP1s of polynomial size. We cannot answer that question but we investigate in Section 2 the expressive power of the different restrictions of OR-BP1s in order to obtain more knowledge about their structural properties. We present an exponential lower bound for nondeterministic tree-driven BP1s for a function even representable by deterministic BP1s of polynomial size.

For a lot of restricted variants of branching programs exponential lower bounds are known. Nevertheless, the proof of exponential lower bounds on the size of BDD models for *natural* functions is not always an easy task.

Definition 8. *Integer multiplication is the Boolean function $MULT_n: \{0,1\}^{2n} \rightarrow \{0,1\}^{2n}$ that computes the product of two n -bit integers. That is, $MULT_n(x, y) = z_{2n-1} \dots z_0$ where $x = x_{n-1} \dots x_0$ and $y = y_{n-1} \dots y_0$ and $xy = z = z_{2n-1} \dots z_0$. $MULT_{i,n}$ computes the i th bit of $MULT_n$.*

For OBDDs Bryant [8] has presented an exponential lower bound of size $2^{n/8}$ for $MULT_{n-1,n}$. But people working on OBDDs agree on the conjecture that the OBDD size is at least of order 2^n . From the proof of Bryant's lower bound for OBDDs it follows by a simple communication complexity argument that $MULT_{n-1,n}$ cannot be computed in polynomial size by k -OBDDs, which consist of k layers of OBDDs respecting the same ordering, [3] or the various nondeterministic OBDDs [9]. Incorporating Ramsey theoretic arguments of Alon and Maass [2] and using the rank method of communication complexity Gergov [9] extends the lower bound to arbitrary linear-length oblivious BPs. It took quite a long time until Ponzio [16] was able to prove an exponential lower bound of size $2^{\Omega(n^{1/2})}$ for $MULT_{n-1,n}$ for BP1s. He doubts that $2^{\Theta(n^{1/2})}$ is the true read-once complexity of $MULT_{n-1,n}$ but the counting technique used in his proof seems limited to this lower bound. Until now an exponential lower bound on the size of $MULT_{n-1,n}$ for a nondeterministic nonoblivious branching program model is unknown. For the lower bound technique based on the rectangle method due to Borodin, Razborov, and Smolensky [6] we have to be able to count the number of 1-inputs which seems to be difficult for $MULT_{n-1,n}$. In Section 3, we present an exponential lower bound for $MULT_{n-1,n}$ on the size of nondeterministic tree-driven BP1s. From this result we obtain more knowledge about the structure of $MULT_n$ which seems to be necessary to improve the lower bounds.

Figure 1 summarizes the results (for more details see Section 2 of this paper and Section 4 of [5]). For a branching program model M we denote by $P(M)$ the class of all (sequences of) Boolean functions representable by polynomial size branching programs of type M . Solid arrows indicate inclusions and slashes through the lines proper inclusions. A dotted line between two classes means

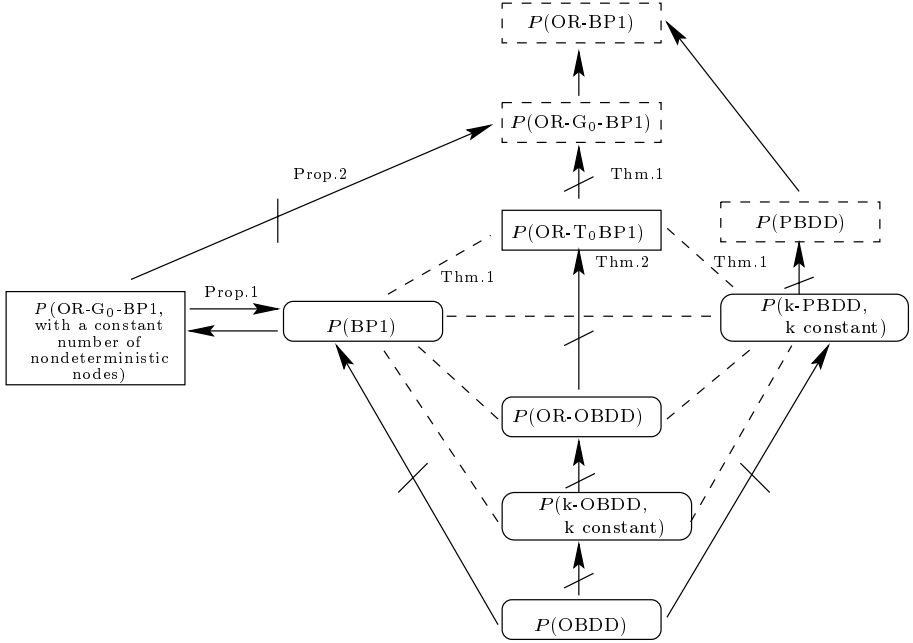


Fig. 1. The complexity landscape for nondeterministic (ordered) read-once branching programs and the classification of $MULT_{n-1,n}$.

that these classes are incomparable. $P(M)$ surrounded by an oval or an rectangle means $MULT_{n-1,n} \notin P(M)$. The ovals indicate known results while the rectangles indicate our new results. A dotted rectangle means that it is unknown whether the class contains $MULT_{n-1,n}$. The numbers in the figure refer to the results of this paper.

2 Restricted Models of Nondeterministic Read-Once Branching Programs

Sauerhoff [17] has asked how much nondeterminism is required to exploit the full power of a computation model and how the complexity of concrete problems depends on the amount of available nondeterminism. He has investigated these questions for OR-OBDDs and has proved that the requirement to test all nondeterministic nodes at the top, i.e., at the beginning of the computation, might blow-up the size exponentially. In order to prove an exponential lower bound for parity read-once branching programs Savický and Sieling [18] have recently presented a hierarchy result for read-once branching programs with restricted parity nondeterminism. Only at the top of the BDD parity nodes are allowed. Their result also holds (and has been stated by the authors) for OR-BP1s. For nondeterministic graph-driven read-once branching programs there cannot exist such a hierarchy result.

Proposition 1. *If a function f_n on n Boolean variables is representable in polynomial size by nondeterministic graph-driven BP1s with a constant number of nondeterministic nodes, f_n is contained in $P(\text{BP1})$.*

Sketch of proof. Using the synthesis algorithm of Sieling and Wegener [19] which combines two functions represented by G_0 -BP1s by a binary operation we can construct a deterministic G_0 -BP1 for f of polynomial size. $\square$

The function 1-VECTOR_n is defined on an $n \times n$ Boolean matrix X and outputs 1 iff the matrix X contains an odd number of ones and a row consisting of ones only or an even number of ones and a column consisting of ones only.

Proposition 2. *The function 1-VECTOR_n on n^2 Boolean variables can be represented by OR-OBDDs of size $O(n^3)$ and by OR-BP1s of size $O(n^2)$ with one nondeterministic node but for OR- G_0 -BP1s with a constant number of nondeterministic nodes the size is $2^{\Omega(n^{1/2})}$.*

Proof. Nondeterministic OBDDs are a restricted variant of nondeterministic tree-driven BP1s. It is easy to see that the function 1-VECTOR_n can be represented by OR-OBDDs with $O(n)$ nondeterministic nodes in size $O(n^3)$. We can guess the row or the column consisting of ones only and check whether the number of ones in the matrix is odd or even. The size does not depend on the choice of the variable ordering.

Bollig and Wegener [5] have shown that 1-VECTOR_n can be represented by 2-PBDDs of size $O(n^2)$. Obviously, PBDDs are very restricted OR-BP1s with nondeterministic nodes only at the top of the BDD. Furthermore, Bollig and Wegener have proved that deterministic BP1s need size $2^{\Omega(n^{1/2})}$. Our result follows from the proof of Proposition 1. $\square$

Unfortunately, we are not able to prove whether there is a sequence of functions $f_n : \{0, 1\}^n \rightarrow \{0, 1\}$ which can be represented by OR-BP1s of polynomial size but for which OR- G_0 -BP1s without restriction of the number of nondeterministic nodes require exponential size. But we conjecture that there exists such a function. The situation is different for OR- G_0 -BP1 and OR- T_0 -BP1.

Sieling and Wegener [19] have shown that the *hidden weighted bit function* HWB which computes x_{sum} where sum is the number of ones in the input needs deterministic tree-driven BP1s of exponential size but has polynomial-size deterministic graph-driven BP1s. Now, we prove that also in the nondeterministic case the expressive power of the two models is different. Using communication complexity Hromkovič and Sauerhoff [12] have presented an exponential lower bound of $2^{\Omega(n)}$ on the size of OR-OBDDs for the function *monochromatic rows or columns* which is defined in the following way. Let X be an $n \times n$ Boolean matrix and z be a Boolean variable. Then

$$\text{MRC}_n(X) := \left(z \wedge \bigwedge_{1 \leq i \leq n} (x_{i,1} \equiv \dots \equiv x_{i,n}) \right) \vee \left(\bar{z} \wedge \bigwedge_{1 \leq i \leq n} (x_{1,i} \equiv \dots \equiv x_{n,i}) \right).$$

Here, we investigate a very similar function $\text{MRC}_n^* : \{0, 1\}^{n^2} \rightarrow \{0, 1\}$ which is only defined on an $n \times n$ Boolean matrix X by

$$\text{MRC}_n^*(X) := \bigwedge_{1 \leq i \leq n} (x_{i,1} \equiv \dots \equiv x_{i,n}) \vee \bigwedge_{1 \leq i \leq n} (x_{1,i} \equiv \dots \equiv x_{n,i}).$$

We prove an exponential lower bound on the OR-OBDD size for MRC_n^* by reducing the *equality function* $\text{EQ}_{n-1} : \{0, 1\}^{n-1} \times \{0, 1\}^{n-1} \rightarrow \{0, 1\}$ to MRC_n^* . Using the fact that the nondeterministic communication complexity of EQ_{n-1} is $n - 1$ it follows that $\text{MRC}_n^* \notin P(\text{OR-OBDD})$. (See, e.g., [11] and [14] for the theory of communication complexity.)

Theorem 1. *There exists a function f_n on n^3 Boolean variables which needs exponential size for OR- T_0 -BP1s but is contained in $P(\text{BP1})$ and $P(2\text{-PBDD})$.*

Proof. The function $f_n : \{0, 1\}^{n^3} \rightarrow \{0, 1\}$ is defined as disjunction of n disjoint copies of MRC_n^* . Let X_i , $1 \leq i \leq n$, be an $n \times n$ Boolean matrix and

$$f_n(X_1, \dots, X_n) := \text{OR}_n(\text{MRC}_n^*(X_1), \dots, \text{MRC}_n^*(X_n)).$$

The proof method is the following one. We assume that f_n has nondeterministic tree-driven BP1s of polynomial size with respect to a tree ordering T_0 . In T_0 there exists a path from the source to the sink which contains only $O(\log n)$ branching nodes, i.e., nodes with different 0- and 1-successor. Fixing the variables labelling these branching nodes in an appropriate way the result is a subfunction of f_n which has to be represented by a so-called nondeterministic list-driven BP1, i.e., a nondeterministic OBDD. If for all subfunctions resulting from f_n by fixing $O(\log n)$ variables by constants the size of nondeterministic OBDDs is exponential, there is a contradiction and we are done.

For any subfunction resulting from f_n by fixing $O(\log n)$ variables by constants there are $n - o(n)$ Boolean matrices X_i , $1 \leq i \leq n$, for which all variables are free, i.e., all n^2 Boolean variables of X_i are not replaced by constants. We choose one of these Boolean matrices X_i and fix all other variables not belonging to X_i in such a way that the resulting subfunction of f_n equals $\text{MRC}_n^*(X_i)$. Now, we use the above mentioned lower bound.

For the first upper bound we construct a BP1 for $f_n(X_1, \dots, X_n)$ of size $O(n^3)$. First, OR_n is represented on the pseudo variables $y_1, \dots, y_n$ by an OBDD of size n . Afterwards, each y_i -node is replaced by a BP1 for MRC_n^* on the X_i -variables. In order to describe the BP1 for MRC_n^* we use an auxiliary tree ordering T_0 . The tree ordering T_0 for MRC_n^* is defined in the following way. We start to test the variables according to a rowwise variable ordering. If the first row contains only 0-entries or only 1-entries, we can proceed with a rowwise variable ordering, otherwise we continue with a columnwise ordering. The width of T_0 is bounded above by $2n$. It is not difficult to see that the size of the tree ordering T_0 as well as the size of the T_0 -BP1 for MRC_n^* is $O(n^2)$.

Now, we prove an upper bound of $O(n^3)$ for the 2-PBDD size of f_n . The first part checks whether there exists a matrix with monochromatic rows. All X_i -variables, $1 \leq i \leq n$, are tested one after another in a rowwise variable ordering. The second part uses a columnwise variable ordering and tests whether there is a matrix consisting of monochromatic columns. $\square$

Proposition 2 and the proof of Theorem 1 also show that the class of functions representable by deterministic tree-driven BP1 and $P(\text{OR-OBDD})$ are incomparable. Furthermore, it is not difficult to see that $P(k\text{-OBDD})$, k constant, is a proper subclass of $P(\text{OR-OBDD})$ [5]. Therefore, all functions representable by polynomial size $k\text{-OBDDs}$, k constant, can also be represented by $\text{OR-}T_0\text{-BP1s}$ of polynomial size. Bollig and Wegener [5] have shown that there are functions in $P(2\text{-OBDD})$ which cannot be represented by $k\text{-PBDDs}$, k constant, of polynomial size. If we relax the restriction for $\text{OR-}T_0\text{-BP1s}$ that the deterministic variables have to be tested according to a tree ordering to the requirement that the labels of nondeterministic and deterministic nodes respect a tree ordering, we obtain a BDD model which can represent all functions of $P(\text{PBDD})$ in polynomial size. But until now no function with polynomial size for OR-BP1s but exponential size for PBDDs with a polynomial number of parts is known.

3 An Exponential Lower Bound for Multiplication

We prove that nondeterministic tree-driven read-once branching programs computing integer multiplication require size $2^{\Omega(n/\log n)}$. This is the first nontrivial lower bound for multiplication on nondeterministic branching programs that are not oblivious.

Lemma 1. *The nondeterministic communication complexity of the problem to decide for l -bit numbers x (given to Alice) and y (given to Bob) whether $|x|+|y| \geq 2^l - c$, where c is a constant of length l with $o(l)$ ones, is $\Omega(l)$.*

Theorem 2. *The size of nondeterministic tree-driven read-once branching programs representing $\text{MULT}_{n-1,n}$ is $2^{\Omega(n/\log n)}$.*

Sketch of proof. Using the proof method of Theorem 1 we show that for each replacement of $O(\log n)$ variables by arbitrary constants we find a subfunction of $\text{MULT}_{n-1,n}$ which essentially equals the computation of the problem from Lemma 1. For this we use the ideas of Bryant's proof [8] but for our case we need some more arguments to limit the influence of the already fixed variables.

We consider an arbitrary subset of $O(\log n)$ variables and an assignment of the variables to constants. Let C_x and C_y be the sets of indices of these x - and y -variables. Variables x_j , $j \notin C_x$, and y_j , $j \notin C_y$, are called free. Let c be the result of MULT_n if we additionally set all free variables to 0 and let C be the set of indices of the 1-bits of c . Obviously $|C| = O(\log^2 n)$.

First, we are looking for a sequence of indices $j, j+1, \dots, j+l-1$ of maximal length such that the input variables $x_j, \dots, x_{j+l-1}$ and $y_{n-1-j-l+1}, \dots, y_{n-1-j}$

are free. Using the pigeonhole principle we estimate below the length of such a sequence by $l = \Omega(n/\log n)$. For the ease of description we assume that l can be divided by 10. Let $X = \{x_j, \dots, x_{j+l-1}\}$ and $Y = \{y_{n-1-j-l+1}, \dots, y_{n-1-j}\}$ be the sets of free variables belonging to such a sequence of maximal length. We choose $X' = \{x_{j+(2/5)l}, \dots, x_{j+(3/5)l-1}\}$. Later we set almost all variables of Y and $X \setminus X'$ to 0 to avoid an undesirable influence of the variables which are not free.

Let π be an arbitrary variable ordering. The top part T of π contains the first $(1/10)l$ X' -variables with respect to π and the bottom part B the other $(1/10)l$ variables. The set of pairs $P = \{(x_{i_1}, x_{i_2}) | x_{i_1} \in T, x_{i_2} \in B\}$ has size $(1/10)l^2$. By a counting argument we find some set $I \subseteq \{j+(2/5)l, \dots, j+(3/5)l-1\}$ and some distance parameter d such that $P' = \{(x_i, x_{i+d}) | i \in I\} \subseteq P$, $|P'| = |I| \geq (1/20)l$, and $\max(I) < \min(I) + d$. We replace the variables in the following way:

- y_k is replaced by 1 for $k = n-1-\max(I)$ and $k = n-1-\max(I)-d$,
- all other free y -variables are replaced by 0,
- x_k is replaced by 1 iff $k \notin I$, $\min(I) < k < \max(I)$, and $k + (n-1-\max(I)) \notin C$,
- $x_{\max(I)+d}$ is replaced by 0 and $x_{\max(I)}$ is replaced by 1 if $n-1 \in C$, otherwise $x_{\max(I)+d}$ and $x_{\max(I)}$ are both replaced by 0,
- all other free x -variables except x_i, x_{i+d} , $i \in I$, are replaced by 0.

All the replacements are possible since all considered variables are free. Due to the lack of space we only give an idea of the effect of these replacements. The output bit of $\text{MULT}_{n-1,n}$ only depends on c and the assignments for $x_{j+(2/5)l}, \dots, x_{j+(3/5)l-1}, y_{n-1-\max(I)}$, and $y_{n-1-\max(I)-d}$. We are left with the situation to add two numbers and the constant c , $\text{MULT}_{n-1,n}$ equals the most significant bit of this sum. Furthermore, the top part T contains exactly one of the free bits of each position. Now the result follows from Lemma 1. $\square$

Since Lemma 1 can be extended to AND- and PARITY-nondeterminism, similar lower bounds for $\text{MULT}_{n-1,n}$ can be proven for AND- T_0 -BP1 and PARITY- T_0 -BP1 in a straightforward way. This is the first nontrivial lower bound even for an important function on nonoblivious branching programs with an unlimited number of parity nodes. Furthermore, an extension of the proof of Theorem 2 shows that all subfunctions of $\text{MULT}_{n-1,n}$ obtained by the replacement of up to $(1-\epsilon)n^{1/2}$ variables by constants have exponential size for nondeterministic tree-driven BP1.

We only want to mention that we obtain similar exponential lower bounds for the arithmetic functions squaring, inversion, and division by so-called read-once projections [4].

Acknowledgement. Thanks to Ingo Wegener for several valuable hints and fruitful discussions.

References

1. Ajtai, M. (1999). A non-linear time lower bound for Boolean branching programs. Proc. of 40th FOCS, 60–70.

2. Alon, N. and Maass, W. (1988). Meanders and their applications in lower bound arguments. *Journal of Computer and System Sciences* 37, 118–129.
3. Bollig, B., Sauerhoff, M., Sieling, D., and Wegener, I. (1993). Read- k times ordered binary decision diagrams. Efficient algorithms in the presence of null chains. Tech. Report 474, Univ. Dortmund.
4. Bollig, B. and Wegener, I. (1998). Completeness and non-completeness results with respect to read-once projections. *Information and Computation* 143, 24–33.
5. Bollig, B. and Wegener, I. (1999). Complexity theoretical results on partitioned (nondeterministic) binary decision diagrams. *Theory of Computing Systems* 32, 487–503.
6. Borodin, A., Razborov, A., and Smolensky, R. (1993). On lower bounds for read- k -times branching programs. *Comput. Complexity* 3, 1–18.
7. Bryant, R. E. (1986). Graph-based algorithms for Boolean manipulation. *IEEE Trans. on Computers* 35, 677–691.
8. Bryant, R. E. (1991). On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Computers* 40, 205–213.
9. Gergov, J. (1994). Time-space trade-offs for integer multiplication on various types of input oblivious sequential machines. *Information Processing Letters* 51, 265–269.
10. Gergov, J. and Meinel, C. (1994). Efficient Boolean manipulation with OBDDs can be extended to FBDDs. *IEEE Trans. on Computers* 43, 1197–1209.
11. Hromkovič, J. (1997). *Communication Complexity and Parallel Computing*. Springer.
12. Hromkovič, J. and Sauerhoff, M. (2000). Communications with restricted nondeterminism and applications to branching program complexity. *Proc. of 17th STACS, Lecture Notes in Computer Science* 1770, 145–156.
13. Jain, J., Abadir, M., Bitner, J., Fussell, D. S., and Abraham, J. A. (1992). Functional partitioning for verification and related problems. *Brown/MIT VLSI Conference*, 210–226.
14. Kushilevitz, E. and Nisan, N. (1997). *Communication Complexity*. Cambridge University Press.
15. Meinel, C. (1990). Polynomial size Ω -branching programs and their computational power. *Information and Computation* 85, 163–182.
16. Ponzio, S. (1998). A lower bound for integer multiplication with read-once branching programs. *SIAM Journal on Computing* 28, 798–815.
17. Sauerhoff, M. (1999). Computing with restricted nondeterminism: The dependence of the OBDD size on the number of nondeterministic variables. *Proc. of 19th FST & TCS, Lecture Notes in Computer Science* 1738, 342–355.
18. Savický, P. and Sieling, D. (2000). A hierarchy result for read-once branching programs with restricted parity nondeterminism. Preprint.
19. Sieling, D. and Wegener, I. (1995). Graph driven BDDs - a new data structure for Boolean functions. *Theoretical Computer Science* 141, 283–310.
20. Thathachar, J. (1998). On separating the read- k -times branching program hierarchy. *Proc. of 30th Ann. ACM Symposium on Theory of Computing (STOC)*, 653–662.
21. Wegener, I. (1987). *The Complexity of Boolean Functions*. Wiley-Teubner.
22. Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications. In print.

Expressiveness of Updatable Timed Automata

P. Bouyer, C. Dufourd, E. Fleury, and A. Petit

LSV, UMR 8643, CNRS & ENS de Cachan,
61 Av. du Président Wilson, 94235 Cachan cedex, France
{bouyer, dufourd, fleury, petit}@lsv.ens-cachan.fr

Abstract. Since their introduction by ALUR and DILL, timed automata have been one of the most widely studied models for real-time systems. The syntactic extension of so-called updatable timed automata allows more powerful updates of clocks than the reset operation proposed in the original model.

We prove that any language accepted by an updatable timed automaton (from classes where emptiness is decidable) is also accepted by a “classical” timed automaton. We propose even more precise results on bisimilarity between updatable and classical timed automata.

1 Introduction

Since their introduction by ALUR and DILL [2,3], timed automata have been one of the most studied models for real-time systems (see [4,1,16,8,12,17,13]). In particular numerous works proposed extensions of timed automata [7,10,11].

This paper focuses on one of this extension, the so-called updatable timed automata, introduced in order to model the ATM protocol ABR [9]. Updatable timed automata are constructed with updates of the following forms:

$$x \sim c \mid x \sim y + c \text{ where } x, y \text{ are clocks, } c \in \mathbb{Q}_+ \text{ and } \sim \in \{<, \leq, =, \neq, \geq, >\}$$

In [5], the (un)decidability of emptiness of updatable timed automata has been characterized in a precise way (see Section 2 for detailed results). We address here the open question of the expressive power of updatable timed automata (from decidable classes). We solve completely this problem by proving that any language accepted by an updatable timed automaton is also accepted by a “classical” timed automaton with ε -transitions. In fact, we propose even more precise results by showing that any updatable timed automaton using only deterministic updates is strongly bisimilar to a classical timed automaton and that any updatable timed automaton using arbitrary updates is weakly bisimilar (but not strongly bisimilar) to a classical timed automaton.

The paper is organized as follows. In Section 2, we present updatable timed automata, generalizing classical definitions of ALUR and DILL. Several natural equivalences of updatable timed automata are introduced in Section 3. The bisimulation algorithms are presented in Section 4.

For lack of space, this paper contains only some sketches of proofs. They are available on the technical report [6].

2 Updatable Timed Automata

Timed Words and Clocks

If Z is any set, let Z^* (respectively Z^ω) be the set of *finite* (resp. *infinite*) sequences of elements in Z and let $Z^\infty = Z^* \cup Z^\omega$. We consider as time domain $\mathbb{T}$ the set of non-negative rational $\mathbb{Q}_+$ and Σ as finite set of *actions*. A *time sequence* over $\mathbb{T}$ is a finite or infinite non decreasing sequence $\tau = (t_i)_{i \geq 1} \in \mathbb{T}^\infty$. A *timed word* $\omega = (a_i, t_i)_{i \geq 1}$ is an element of $(\Sigma \times \mathbb{T})^\infty$.

We consider an at most countable set $\mathbb{X}$ of variables, called *clocks*. A clock valuation over $\mathbb{X}$ is a mapping $v : \mathbb{X} \rightarrow \mathbb{T}$ that assigns to each clock a time value. Let $t \in \mathbb{T}$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t, \forall x \in \mathbb{X}$.

Clock Constraints

Given a subset of clocks $X \subseteq \mathbb{X}$, we introduce two sets of clock constraints over X . The most general one, denoted by $\mathcal{C}(X)$, is defined by the following grammar:

$$\varphi ::= x \sim c \mid x - y \sim c \mid \varphi \wedge \varphi \mid \neg \varphi \mid \text{true}, \text{ with } x, y \in X, c \in \mathbb{Q}_+, \sim \in \{<, \leq, =, \neq, \geq, >\}$$

The proper subset $\mathcal{C}_{df}(X)$ of “diagonal-free” constraints in which the comparison between two clocks is not allowed, is defined by the grammar:

$$\varphi ::= x \sim c \mid \varphi \wedge \varphi \mid \neg \varphi \mid \text{true}, \text{ with } x \in X, c \in \mathbb{Q}_+ \text{ and } \sim \in \{<, \leq, =, \neq, \geq, >\}$$

We write $v \models \varphi$ when the clock valuation v satisfies the clock constraint φ .

Updates

An *update* is a function which assigns to each valuation a set of valuations. Here, we restrict ourselves to local updates which are defined in the following way. A *simple update* over a clock z is of one of the two following forms:

$$up ::= z \sim c \mid z \sim y + d, \text{ where } c, d \in \mathbb{Q}_+, y \in \mathbb{X} \text{ and } \sim \in \{<, \leq, =, \neq, \geq, >\}$$

When the operator $\sim$ is the equality ($=$), the update is said to be *deterministic*, *non deterministic* otherwise. Let v be a valuation and up be a simple update over z . A valuation v' is in $up(v)$ if $v'(y) = v(y)$ for any clock $y \neq z$ and if $v'(z) \sim c$ ($v'(z) \sim v(y) + d$ resp.) if $up = z \sim c$ ($up = z \sim y + d$ resp.)

The set $lu(\mathcal{U})$ of local updates generated by a set of simple updates $\mathcal{U}$ is defined as follows. A collection $up = (up_i)_{1 \leq i \leq k}$ is in $lu(\mathcal{U})$ if, for each i , up_i is a simple update of $\mathcal{U}$ over some clock $x_i \in X$ (note that it could happen that $x_i = x_j$ for some $i \neq j$). Let $v, v' \in \mathbb{T}^n$ be two clock valuations. We have $v' \in up(v)$ if and only if, for any i , the clock valuation v'' defined by $v''(x_i) = v'(x_i)$ and $v''(y) = v(y)$ for any $y \neq x_i$ verifies $v'' \in up_i(v)$.

Note that $up(v)$ may be empty. For instance, the local update $(x < 1, x > 1)$ leads to an empty set. But if we take the local update $(x > y, x < 7)$, the value $v'(x)$ has to satisfy : $v'(x) > v(y) \wedge v'(x) < 7$.

For any subset X of $\mathbb{X}$, $\mathcal{U}(X)$ is the set of local updates which are collections of simple updates over clocks of X . In the following, $\mathcal{U}_0(X)$ denotes the set of reset updates. A reset update is an update up such that for every clock valuation v, v' with $v' \in up(v)$ and any clock $x \in X$, either $v'(x) = v(x)$ or $v'(x) = 0$. It is precisely this set of updates which was used in “classical” timed automata [3].

Updatable Timed Automata

An *updatable timed automaton* over $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma, Q, X, T, I, F, R)$, where Σ is a finite alphabet of actions, Q a finite set of states, $X \subseteq \mathbb{X}$ a finite set of clocks, $T \subseteq Q \times [\mathcal{C}(X) \times \Sigma \cup \{\varepsilon\} \times \mathcal{U}(X)] \times Q$ a finite set of transitions, $I \subseteq Q$ ($F \subseteq Q$, $R \subseteq Q$ resp.) the subset of initial (final, repeated resp.) states.

Let $\mathcal{C} \subset \mathcal{C}(\mathbb{X})$ be a subset of clock constraints and $\mathcal{U} \subset \mathcal{U}(\mathbb{X})$ be a subset of updates, the class $Aut_\varepsilon(\mathcal{C}, \mathcal{U})$ is the set of all timed automata whose transitions only use clock constraints of $\mathcal{C}$ and updates of $\mathcal{U}$. The usual class of timed automata, defined in [2], is the family $Aut_\varepsilon(\mathcal{C}_{df}(\mathbb{X}), \mathcal{U}_0(\mathbb{X}))$.

A *path* in $\mathcal{A}$ is a finite or an infinite sequence of consecutive transitions:

$$P = q_0 \xrightarrow{\varphi_1, a_1, up_1} q_1 \xrightarrow{\varphi_2, a_2, up_2} q_2 \dots, \text{ where } (q_{i-1}, \varphi_i, a_i, up_i, q_i) \in T, \forall i > 0$$

The path is said *accepting* if $q_0 \in I$ and *either* it is finite and it ends in an final state, *or* it is infinite and passes infinitely often through a repeated state. A *run* of the automaton through the path P is a sequence of the form:

$$\langle q_0, v_0 \rangle \xrightarrow[t_1]{\varphi_1, a_1, up_1} \langle q_1, v_1 \rangle \xrightarrow[t_2]{\varphi_2, a_2, up_2} \langle q_2, v_2 \rangle \dots$$

where $\tau = (t_i)_{i \geq 1}$ is a time sequence and $(v_i)_{i \geq 0}$ are clock valuations such that $\forall x \in \mathbb{X}, v_0(x) = 0$ and $\forall i \geq 1, v_{i-1} + (t_i - t_{i-1}) \models \varphi_i$ and $v_i \in up_i(v_{i-1} + (t_i - t_{i-1}))$. Remark that any set $up_i(v_{i-1} + (t_i - t_{i-1}))$ of a run is non empty.

The label of the run is the sequence $(a_1, t_1)(a_2, t_2) \dots \in ((\Sigma \cup \{\varepsilon\}) \times \mathbb{T})^\infty$. The timed word associated with this sequence is $w = (a_{i_1}, t_{i_1})(a_{i_2}, t_{i_2}) \dots$ where $a_{i_1} a_{i_2} \dots$ is the sequence of actions which are in Σ (i.e. distinct from ε). If the path P is accepting then the timed word w is accepted by the timed automaton.

About Decidability of Updatable Timed Automata

For verification purposes, a fundamental question is to know if the emptiness of (the language accepted by) an updatable timed automaton is decidable or not. The paper [5] proposes a precise characterization which is summarized in the picture below. Note that decidability can depend on the set of clock constraints that are used – diagonal-free or not – which makes an important difference with “classical” timed automata for which it is well known that these two kinds of constraints are equivalent. The technique proposed in [5] shows that all the decidability cases are PSPACE-complete.

		diagonal-free clock constraints	general clock constraints
Deterministic updates	$x := c ; x := y$	DECIDABLE	DECIDABLE
	$x := y + c, c \in \mathbb{Q}^+$	DECIDABLE	UNDECIDABLE
	$x := y + c, c \in \mathbb{Q}^-$	UNDECIDABLE	UNDECIDABLE
Non deterministic updates	$x :< c, c \in \mathbb{Q}^+$	DECIDABLE	DECIDABLE
	$x :> c, c \in \mathbb{Q}^+$	DECIDABLE	UNDECIDABLE
	$x :< y + c, c \in \mathbb{Q}^+$	DECIDABLE	UNDECIDABLE
	$x :> y + c, c \in \mathbb{Q}^+$	DECIDABLE	UNDECIDABLE

The present paper addresses the natural question of the exact expressive power of the decidable classes. To solve this problem, we first introduce natural and classical equivalences between updatable timed automata.

3 Some Equivalences of Updatable Timed Automata

Language Equivalence

Two updatable timed automata are *language-equivalent* if they accept the same timed language. By extension, two families Aut_1 and Aut_2 are said to be equivalent if any automaton of one of the families is equivalent to one automaton of the other. We write $\equiv_\ell$ in both cases. For instance, $Aut_\varepsilon(\mathcal{C}_{df}(\mathbb{X}), \mathcal{U}_0(\mathbb{X})) \equiv_\ell Aut_\varepsilon(\mathcal{C}(\mathbb{X}), \mathcal{U}_0(\mathbb{X}))$, (see e.g. [7]).

Bisimilarity

Bisimilarity [15,14] is stronger than language equivalence. It defines a step by step correspondence between two transition systems. Two labelled transition systems $\mathcal{T} = (S, S_0, E, (\xrightarrow{e})_{e \in E})$ and $\mathcal{T}' = (S', S'_0, E, (\xrightarrow{e})_{e \in E})$ are *bisimilar* whenever there exists a relation $\mathcal{R} \subseteq S \times S'$ which meets the following conditions:

$$\begin{aligned} \text{INITIALIZATION : } & \begin{cases} \forall s_0 \in S_0, \exists s'_0 \in S'_0 \text{ such that } s_0 \mathcal{R} s'_0 \\ \forall s'_0 \in S'_0, \exists s_0 \in S_0 \text{ such that } s_0 \mathcal{R} s'_0 \end{cases} \\ \text{PROPAGATION : } & \begin{cases} \text{if } s_1 \mathcal{R} s'_1 \text{ and } s_1 \xrightarrow{e} s_2 \text{ then there exists } s'_2 \in S' \\ \quad \text{such that } s'_1 \xrightarrow{e} s'_2 \text{ and } s_2 \mathcal{R} s'_2 \\ \text{if } s_1 \mathcal{R} s'_1 \text{ and } s'_1 \xrightarrow{e} s'_2 \text{ then there exists } s_2 \in S \\ \quad \text{such that } s_1 \xrightarrow{e} s_2 \text{ and } s_2 \mathcal{R} s'_2 \end{cases} \end{aligned}$$

Strong and Weak Bisimilarity

Timed transition systems - Each updatable timed automaton $\mathcal{A} = (\Sigma, Q, X, T, I, F, R)$ in $Aut_\varepsilon(\mathcal{C}(\mathbb{X}), \mathcal{U}(\mathbb{X}))$ defines a timed transition system $\mathcal{T}_{\mathcal{A}} = (S, S_0, E, (\xrightarrow{e})_{e \in E})$ as follows :

- $S = Q \times \mathbb{T}^X$, $S_0 = \{\langle q, v \rangle \mid q \in I \text{ and } \forall x \in X, v(x) = 0\}$, $E = \Sigma \cup \{\varepsilon\} \cup \mathbb{Q}_+$
- $\forall a \in \Sigma \cup \{\varepsilon\}$, $\langle q, v \rangle \xrightarrow{a} \langle q', v' \rangle$ iff $\exists (q, \varphi, a, up, q') \in T$ s.t. $v \models \varphi$ and $v' \in up(v)$
- $\forall d \in \mathbb{Q}_+$, $\langle q, v \rangle \xrightarrow{d} \langle q', v' \rangle$ iff $q = q'$ and $v' = v + d$

When ε is considered as an invisible action, each updatable timed automaton $\mathcal{A}$ in $Aut_\varepsilon(\mathcal{C}(\mathbb{X}), \mathcal{U}(\mathbb{X}))$ defines another transition system $\mathcal{T}'_{\mathcal{A}} = (S, S_0, E', (\xrightarrow{e})_{e \in E})$ as follows:

- $S = Q \times \mathbb{T}^X$, $S_0 = \{\langle q, v \rangle \mid q \in I \text{ and } \forall x \in X, v(x) = 0\}$, $E' = \Sigma \cup \mathbb{Q}_+$
- $\forall a \in \Sigma$, $\langle q, v \rangle \xrightarrow{a} \langle q', v' \rangle$ iff $\langle q, v \rangle \xrightarrow{\varepsilon^*} \xrightarrow{a} \xrightarrow{\varepsilon^*} \langle q', v' \rangle$
- $\forall d \in \mathbb{Q}_+$, $\langle q, v \rangle \xrightarrow{d} \langle q', v' \rangle$ iff $\langle q, v \rangle \xrightarrow{\varepsilon^*} \xrightarrow{d_1} \xrightarrow{\varepsilon^*} \dots \xrightarrow{d_k} \xrightarrow{\varepsilon^*} \langle q', v' \rangle$ and $d = \sum_{i=1}^k d_i$

Two bisimilarities for timed automata - Two updatable timed automata $\mathcal{A}$ and $\mathcal{B}$ are *strongly bisimilar*, denoted $\mathcal{A} \equiv_s \mathcal{B}$, if $\mathcal{T}_{\mathcal{A}}$ and $\mathcal{T}_{\mathcal{B}}$ are bisimilar. They are *weakly bisimilar*, denoted $\mathcal{A} \equiv_w \mathcal{B}$, if $\mathcal{T}'_{\mathcal{A}}$ and $\mathcal{T}'_{\mathcal{B}}$ are bisimilar.

Remark 1. Two timed strongly bisimilar automata are obviously weakly bisimilar. If the bisimulation $\mathcal{R}$ preserves the final and repeated states, weakly or strongly bisimilar updatable timed automata recognize the same language.

Let $\mathcal{A}$ a timed automaton and λ be a constant. We denote by $\lambda\mathcal{A}$ the timed automaton in which all the constants which appear are multiplied by the constant λ . The proof of the following lemma is immediate and similar to the one of Lemma 4.1 in [3]. This lemma allows us to treat only updatable timed automata where all constants appearing in the clock constraints and in the updates are integer (and not arbitrary rationals).

Lemma 1. *Let $\mathcal{A}$ and $\mathcal{B}$ be two timed automata and $\lambda \in \mathbb{Q}^+$ be a constant. Then $\mathcal{A} \equiv_w \mathcal{B} \iff \lambda\mathcal{A} \equiv_w \lambda\mathcal{B}$ and $\mathcal{A} \equiv_s \mathcal{B} \iff \lambda\mathcal{A} \equiv_s \lambda\mathcal{B}$*

4 Expressive Power of Deterministic Updates

We first deal with updatable timed automata where only deterministic updates are used. The following theorem is often considered as a “folklore” result.

Theorem 1. *Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a set of clock constraints and let $\mathcal{U} \subseteq lu(\{x := d \mid x \in X \text{ and } d \in \mathbb{Q}^+\} \cup \{x := y \mid x, y \in X\})$. Let $\mathcal{A}$ be in $Aut_\varepsilon(\mathcal{C}, \mathcal{U})$. Then there exists $\mathcal{B}$ in $Aut_\varepsilon(\mathcal{C}(X), \mathcal{U}_0(X))$ such that $\mathcal{A} \equiv_s \mathcal{B}$.*

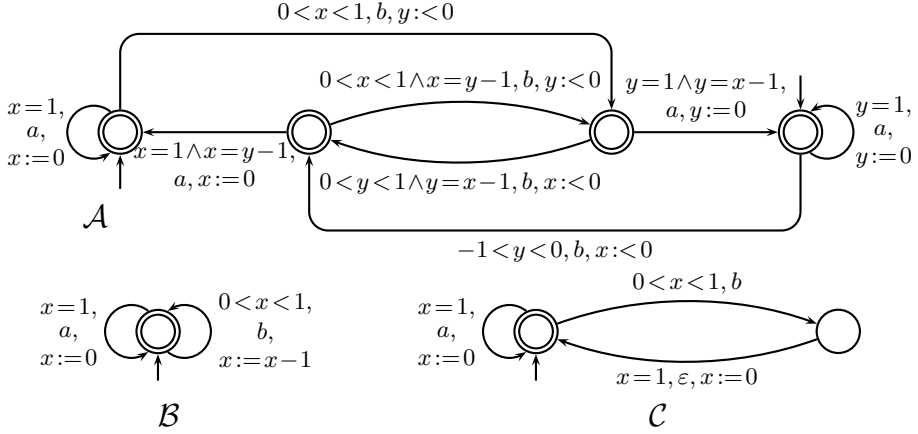
The next theorem is close to the previous one. Note nevertheless that this theorem becomes false if we consider arbitrary clock constraints, since as we recalled in section 2, the corresponding class is undecidable.

Theorem 2. *Let $\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be a set of diagonal-free clock constraints. Let $\mathcal{U} \subseteq lu(\{x := d \mid x \in X \text{ and } d \in \mathbb{Q}^+\} \cup \{x := y + d \mid x, y \in X \text{ and } d \in \mathbb{Q}^+\})$. Let $\mathcal{A}$ be in $Aut_\varepsilon(\mathcal{C}, \mathcal{U})$. Then there exists $\mathcal{B}$ in $Aut_\varepsilon(\mathcal{C}_{df}(X), \mathcal{U}_0(X))$ such that $\mathcal{A} \equiv_s \mathcal{B}$.*

5 Expressive Power of Non Deterministic Updates

In the case of non deterministic updates, we first show that it is hopeless to obtain strong bisimulation with classical timed automata. To this purpose, let us consider the automaton $\mathcal{C}$ of Figure 1. It has been proved in [7] that there is no classical timed automaton without ε -transitions that recognize the same language than $\mathcal{C}$.

Now, it is not difficult to prove that the automaton $\mathcal{C}$ recognizes the same language than the automaton $\mathcal{B}$ and that $\mathcal{B}$ recognizes itself the same language than $\mathcal{A}$. If $\mathcal{A}$ was strongly bisimilar to some automaton $\mathcal{D}$ of $Aut_\varepsilon(\mathcal{C}(X), \mathcal{U}_0(X))$, this automaton $\mathcal{D}$ would not contain any ε -transition (since $\mathcal{A}$ does not contain such transition). Hence $L(\mathcal{D})$ would be equal to $L(\mathcal{A}) = L(\mathcal{C})$, in contradiction with the result of [7] recalled above. Since $\mathcal{A}$ belongs to the class $Aut_\varepsilon(\mathcal{C}(X), \mathcal{U}_1(X))$ (where $\mathcal{U}_1(X)$ denotes the set of updates corresponding to the cells labelled “decidable” in the “general clock constraints” column in tabular of Section 2), we thus have proved:

Fig. 1. Timed automata $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$

Proposition 1. $Aut_\varepsilon(\mathcal{C}(\mathbb{X}), \mathcal{U}_1(\mathbb{X})) \not\equiv_s Aut_\varepsilon(\mathcal{C}(\mathbb{X}), \mathcal{U}_0(\mathbb{X}))$

We now focus on weak bisimilarity. As it will appear, the construction of an automaton of $Aut(\mathcal{C}(\mathbb{X}), \mathcal{U}_0(\mathbb{X}))$ weakly bisimilar to a given automaton of $Aut(\mathcal{C}(\mathbb{X}), \mathcal{U}_1(\mathbb{X}))$ is rather technical. As we recalled in Section 2, the decidable classes of updatable timed automata depend on the set of clock constraints that are used. We consider first the case of diagonal-free clock constraints.

We first propose a normal form for diagonal-free updatable automata. Let $(c_x)_{x \in X}$ be a family of constants of $\mathbb{N}$. In what follows we will restrict ourselves to the clock constraints $x \sim c$ where $c \leq c_x$. We define: $\mathcal{I}_x = \{[d; d+1[\mid 0 \leq d < c_x\} \cup \{[d \mid 0 \leq d \leq c_x\} \cup \{c_x, \infty\}$

A clock constraint φ is said to be *total* if φ is a conjunction $\bigwedge_{x \in X} I_x$ where for each clock x , I_x is an element of $\mathcal{I}_x$. Any diagonal free clock constraint bounded by the constants $(c_x)_{x \in X}$ is equivalent to a disjunction of total clock constraints.

We define $\mathcal{I}'_x = \{[d; d+1[\mid 0 \leq d < c_x\} \cup \{c_x, \infty\}$. An update up_x is *elementary* if it is of one of the two following forms:

- $x := c$ or $x \in I'_x$ with $I'_x \in \mathcal{I}'_x$,
- $\bigwedge_{y \in H} x \sim y + c \wedge x \in I'_x$ with $\sim \in \{=, <, >\}$, $I'_x \in \mathcal{I}'_x$ and $\forall y \in H, c_x \leq c_y + c$.

An elementary update $((\bigwedge_{y \in H} x \sim y + c) \wedge x \in I'_x)$ is *compatible* with a total guard $\bigwedge_{x \in X} I_x$ if, for any $y \in H$, $I_y + c \subseteq I'_x$. By applying classical rules of propositional calculus and splitting the transitions, we obtain the following normal form for diagonal-free updatable timed automata.

Proposition 2. Any diagonal-free updatable timed automaton from $Aut_\varepsilon(\mathcal{C}_{df}(X), \mathcal{U}(X))$ is strongly bisimilar to a diagonal-free updatable timed automaton from $Aut_\varepsilon(\mathcal{C}_{df}(X), \mathcal{U}(X))$ in which for any transition (p, φ, a, up, q) it holds:

- φ is a total guard
- $up = \bigwedge_{x \in X} up_x$ with for any x , up_x is an elementary update compatible with φ

Construction for Diagonal-Free Updatable Timed Automata

We can now state our main result concerning updatable diagonal-free timed automata:

Theorem 3. *Let $\mathcal{C} \subseteq \mathcal{C}_{df}(X)$ be a set of diagonal-free clock constraints. Let $\mathcal{U} \subseteq \mathcal{U}(X)$ be a set of updates. Let $\mathcal{A}$ be in $\text{Aut}(\mathcal{C}, \mathcal{U})$. Then there exists $\mathcal{B}$ in $\text{Aut}_\varepsilon(\mathcal{C}_{df}(X), \mathcal{U}_0(\mathbb{X}))$ such that $\mathcal{A} \equiv_w \mathcal{B}$. In particular $\mathcal{A}$ and $\mathcal{B}$ accept the same timed language.*

Proof (Sketch of proof). Thanks to Lemma 1 and Proposition 2, we assume that all the constants appearing in $\mathcal{A}$ are integers and that $\mathcal{A}$ is in normal form for some constants $(c_x)_{x \in X}$. For each clock x , we denote by $\mathcal{I}_x''$ the set of intervals $\{]c; c + 1[\mid 0 \leq c < c_x\}$.

A clock x is said *fixed* if the last update of x was of the form either $x := c$ or $(\bigwedge_{y \in H} x := y + c) \wedge x \in I'_x$ where all the clocks of H were fixed themselves. A clock which is not fixed is said *floating*.

The transformation algorithm consists in constructing (a lot of) copies of the original automaton $\mathcal{A}$, adding suitable clocks, transforming the existing transitions and finally adding ε -transitions going from one copy to another.

Duplication of the initial automaton - For each subset $Y \subseteq X$, for each tuple $(I_y)_{y \in Y}$ with $I_y \in \mathcal{I}_y''$, for each partial preorder $\prec$ defined on Y and for each subset $Z \subseteq Y$, we consider a copy of $\mathcal{A}$, denoted by $\mathcal{A}_{(I_y)_{y \in Y}, \prec, Z}$. Intuitively, each clock $y \in Y$ will be floating and with I_y as set of possible values. The preorder $\prec$ corresponds to the partial order between the fractional parts of the clocks of Y . The role of Z will be explained below.

Keeping in mind the fractional part of the clocks - We associate with each clock x an other clock z_x representing the fractional part of x . In an automaton $\mathcal{A}_{(I_y)_{y \in Y}, \prec, Z}$, we need to force the fractional part of any clock x to stay in $[0; 1[$. If a fractional part reaches the value 1, then either the clock is in Y and we will change of automaton (see below) or the clock is not in Y and the fractional part has to be reset to 0. To this purpose, we add to this automaton:

- on each transition, the clock constraint $\bigwedge_{x \in X} (z_x < 1)$
- on each state r , for each clock $x \in X \setminus Y$, a loop $(r, z_x = 1, \varepsilon, z_x := 0, r)$

Erasing some transitions - Since in an automaton $\mathcal{A}_{(I_y)_{y \in Y}, \prec, Z}$, a clock $y \in Y$ will always verify $y \in I_y$, a total clock constraint $\varphi \wedge \bigwedge_x x \in I'_x$ can be satisfied only if $I'_y = I_y$ for all $y \in Y$. Therefore, we erase all the transitions with clock constraints which do not have this property.

Modification of the updates - We consider a copy $\mathcal{A}_{(I_y)_{y \in Y}, \prec, Z}$ and a transition (q, φ, a, up, q') inside this copy. This transition will be replaced by another transition $(q, \varphi, a, \widehat{up}, \widehat{q}')$ from $\mathcal{A}_{(I_y)_{y \in Y}, \prec, Z}$ to another automaton $\mathcal{A}_{(\widehat{I}_y)_{y \in \widehat{Y}}, \widehat{\prec}, \widehat{Z}}$

(which can be possibly $\mathcal{A}_{(I_y)_{y \in Y, \prec, Z}}$ itself) and where $\hat{q}'$ is the copy of q' in the new automaton. The elements $\hat{Y}$, $(I_y)_{y \in Y}$, $\hat{\prec}$ and $\hat{up}$ are constructed inductively by considering one after the other the updates up_x involved in up (the order in which the updates are treated is irrelevant). The new update $\hat{up}$ will be only constituted of updates of the form $x := c$ or $x := y + c$. Initially, we set $\hat{Y} = Y$, $\hat{I}_y = I_y$ for all $y \in Y$, $\hat{\prec} = \prec$, $\hat{up} = true$ and $\hat{Z} = Z$.

Before listing the different updates, let us explain the role of the set Z . Assume that a clock x is updated by an instruction $x < y + c$ where y is floating. Then the clock x is added to the set of floating clocks. Since we do not want to use anymore non deterministic updates, we update the fractional part z_x to 0, $z_x := 0$. But we need to keep the current value of z_y in order to ensure that z_x , which has to be smaller than z_y , will not reach 1 before z_y . Of course, it can be checked easily if y is not updated but if it is the case, we do not have any way to verify this fact. Therefore, in such a case, we add the clock x to the set Z and we use a new clock w_x to keep in mind the current value of z_y : $w_x := z_y$. The required property is then verified by the condition $w_x \geq 1$.

- if up_x is equal to $x := c$ then we just have to consider x as fixed:
 - $\hat{Y} \leftarrow \hat{Y} \setminus \{x\}$, $\hat{Z} \leftarrow \hat{Z} \setminus \{x\}$, $\hat{up} \leftarrow \hat{up} \wedge x := c \wedge z_x := 0$
- if up_x is equal to $\bigwedge_{y \in H} x := y + c \wedge x \in I'_x$ then :
 1. if I'_x is bounded, then we write H as the disjoint union of $H_1 = H \cap Y$ and $H_2 = H \setminus Y$. We distinguish two cases:
 - a) if $H_1 = \emptyset$, then:
 - $\hat{Y} \leftarrow \hat{Y} \setminus \{x\}$, $\hat{Z} \leftarrow \hat{Z} \setminus \{x\}$
 - $\hat{up} \leftarrow \hat{up} \wedge \bigwedge_{y \in H} (x := y + c \wedge z_x := z_y)$
 - b) if $H_1 \neq \emptyset$, then:
 - $\hat{Y} \leftarrow \hat{Y} \cup \{x\}$, $\hat{I}_x \leftarrow I'_x$, $\hat{Z} \leftarrow \hat{Z} \cup \{x\}$
 - for each $y \in H_1$, $x \hat{\prec} y$ and $y \hat{\prec} x$
 - $\hat{up} \leftarrow \hat{up} \wedge \bigwedge_{y \in H_2} (x := y + c \wedge z_x := z_y) \wedge \bigwedge_{y \in H_2} (w_x := z_y)$ if $H_2 \neq \emptyset$; $\hat{up} \leftarrow \hat{up} \wedge (z_x := 0)$ if $H_2 = \emptyset$
 2. if I'_x is non bounded, then we write H as the disjoint union of $H_1 = H \cap Y$ and $H_2 = H \setminus Y$. We distinguish two cases:
 - a) if $H_1 = \emptyset$, then:
 - $\hat{Y} \leftarrow \hat{Y} \setminus \{x\}$, $\hat{Z} \leftarrow \hat{Z} \setminus \{x\}$
 - $\hat{up} \leftarrow \hat{up} \wedge \bigwedge_{y \in H} (x := y + c \wedge z_x := z_y)$
 - b) if $H_1 \neq \emptyset$, then:
 - $\hat{Y} \leftarrow \hat{Y} \cup \{t\} \setminus \{x\}$ where t is some clock of H_2
 - $\hat{I}_t$ is some tested interval (we test whether the value of t is in some interval $]c; c + 1[$ and the clock t becomes a floating clock in this interval)
 - $\hat{Z} \leftarrow \hat{Z} \cup \{t\} \setminus \{x\}$, for each clock $y \in H_1$, $t \hat{\prec} y$ and $y \hat{\prec} t$
 - $\hat{up} \leftarrow \hat{up} \wedge \bigwedge_{y \in H_2} (x' := y + c) \wedge (x := c_x + 1 \wedge z_x := 0) \wedge (w_t := z_t)$
- if up_x is equal to $\bigwedge_{y \in H} x < y + c \wedge x \in I'_x$ then there are two cases:
 1. if I'_x is bounded, then:
 - $\hat{Y} \leftarrow \hat{Y} \cup \{x\}$, $\hat{I}_x \leftarrow I'_x$
 - for all $y \in Y \cap H$, $x \hat{\prec} y$ (but not $y \hat{\prec} x$) is added to $\hat{\prec}$

- $\widehat{Z} \leftarrow \widehat{Z} \setminus \{x\}$ if $H \subseteq Y$; $\widehat{Z} \leftarrow \widehat{Z} \cup \{x\}$ if $H \not\subseteq Y$
 - $\widehat{up} \leftarrow \widehat{up} \wedge (z_x := 0) \wedge \bigwedge_{y \in H \setminus Y} (w_x := z_y)$
2. if I'_x is non bounded, then:
- $\widehat{Y} \leftarrow \widehat{Y} \setminus \{x\}$, $\widehat{up} \leftarrow \widehat{up} \wedge (x := c_x + 1) \wedge (z_x := 0)$
- if up_x is equal to $\bigwedge_y x :> y + c \wedge x : \in I'_x$ then:
1. if I'_x is non bounded, then:
- $\widehat{Y} \leftarrow \widehat{Y} \cup \{x\}$, $\widehat{I}_x \leftarrow I'_x$, $\widehat{Z} \leftarrow \widehat{Z} \setminus \{x\}$
 - $y \hat{<} x$ (but not $x \hat{<} y$) is added to $\hat{<}$, $\widehat{up} \leftarrow \widehat{up} \wedge z_x := z_y$
2. if I'_x is bounded, then:
- $\widehat{Y} \leftarrow \widehat{Y} \setminus \{x\}$, $\widehat{up} \leftarrow \widehat{up} \wedge (x := c_x + 1) \wedge (z_x := 0)$

Adding deterministic updates to go from one copy to another- We consider a particular copy $\mathcal{A}_{(I_y)_{y \in Y, \prec, Z}}$. We add new ε -transitions in order to leave this automaton as soon as some clock y leaves the interval I_y . By definition, the clocks which will first leave I_y belong to the maximal elements of the preorder $\prec$. Therefore, for any subset M of Y such that the elements of M are maximal elements of the preorder $\prec$ and if $y \in M$, $x \prec y$ and $y \prec x$ then $x \in M$, we add a transition from any copy of a state q in $\mathcal{A}_{(I_y)_{y \in Y, \prec, Z}}$ to the copy of q in the automaton $\mathcal{A}_{(I_y)_{y \in Y', \prec', Z'}}$ with $Y' = Y \setminus M$, $\prec' = \prec \cap (Y' \times Y')$, $Z' = Z \setminus M$. This transition is labelled by

$$\bigwedge_{x \in X \cap Z} (z_x \leq 1) \wedge \bigwedge_{x \in X \setminus Z} (z_x < 1) \wedge \bigwedge_{x \in Z \cap M} (w_x \geq 1), \varepsilon, \forall x \in M x := \sup(I_x)$$

where $\sup(I_x)$ is the least upper bound of the interval I_x .

Intuitively it means that the values of some maximal elements have reached the upper bound of their floating interval and thus become fixed.

Now, we just need to define a weak bisimulation $\mathcal{R}$. Roughly, a state of the original timed automaton will be in relation with all the copies of this state (with appropriate valuations). This concludes the proof of Theorem 3.

When we deal with the general updatable timed automata, as we recalled in Section 2, we need to restrict deeply the updates that are used in order to have decidable classes. As states the next theorem, these classes are once again weakly bisimilar to classical timed automata with ε -transitions.

Theorem 4. *Let $\mathcal{C} \subseteq \mathcal{C}(X)$ be a set of general clock constraints and $\mathcal{U} \subseteq lu(\{x := y \mid x, y \in X\} \cup \{x \sim c \mid x \in X, c \in \mathbb{Q}^+ \text{ and } \sim \in \{<, \leq, =\}\})$ be a set of updates. Let $\mathcal{A}$ be in $Aut(\mathcal{C}, \mathcal{U})$. Then there exists $\mathcal{B}$ in $Aut_\varepsilon(\mathcal{C}(X), \mathcal{U}_0(X))$ such that $\mathcal{A} \equiv_w \mathcal{B}$. In particular $\mathcal{A}$ and $\mathcal{B}$ accept the same timed language.*

The proof is quite similar to the one of Theorem 3, and even simpler because there is no non deterministic update allowed and involving two clocks.

6 Conclusion

Our results are summarized in the following tabular (a $\times$ denotes an undecidable case). A cell labelled “(Strongly/Weakly) bisimilar” means that any updatable timed automaton of the class represented by the cell is (strongly/weakly) bisimilar to a “classical” timed automaton with ε -transitions:

		Diagonal-free constraints	General constraints
Determ. updates	$x := c ; x := y$	Strongly bisimilar	Strongly bisimilar
	$x := y + c, c \in \mathbb{Q}^+$	Strongly bisimilar	$\times$
Non Determ. updates	$x < c, c \in \mathbb{Q}^+$	Weakly bisimilar	Weakly bisimilar
	$x > c, c \in \mathbb{Q}^+$	Weakly bisimilar	$\times$
	$x \sim y + c, c \in \mathbb{Q}^+, \sim \in \{<, >\}$	Weakly bisimilar	$\times$

References

1. R. Alur, C. Courcoubetis, and T.A. Henzinger. The observational power of clocks. In *Proc. of CONCUR'94*, LNCS 836, pages 162–177, 1994.
2. R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. of ICALP'90*, LNCS 443, pages 322–335, 1990.
3. R. Alur and D. Dill. A theory of timed automata. *TCS'94*, pages 183–235, 1994.
4. R. Alur, T.A. Henzinger, and M. Vardi. Parametric real-time reasoning. In *Proc. of the 25th ACM STOC*, pages 592–601, 1993.
5. P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable ? In *Proc. of CAV'2000*, LNCS, 2000. To appear.
6. P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Expressiveness of updatable timed automata. Research report, ENS de Cachan, 2000.
7. B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, pages 145–182, 1998.
8. B. Bérard and C. Dufourd. Timed automata and additive clock constraints. Research report LSV-00-4, LSV, ENS de Cachan, 2000.
9. B. Bérard and L. Fribourg. Automatic verification of a parametric real-time program : the ABR conformance protocol. In *Proc. of CAV'99*, LNCS 1633, pages 96–107, 1999.
10. C. Choffrut and M. Goldwurm. Timed automata with periodic clock constraints. Technical Report 99/28, LIAFA, Université Paris VII, 1999.
11. F. Demichelis and W. Zielonka. Controlled timed automata. In *Proc. of CONCUR'98*, LNCS 1466, pages 455–469, 1998.
12. T.A. Henzinger, P. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Software Tools for Technology Transfer*, pages 110–122, 1997. (special issue on Timed and Hybrid Systems).
13. K.G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1:134–152, 1997.
14. R. Milner. *Communication and Concurrency*. Prentice Hall Int., 1989.

15. D. M. Park. Concurrency on automata and infinite sequences. In *CTCS'81*, LNCS 104, pages 167–183, 1981.
16. T. Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In *Proc. of FTRT-FTS*, LNCS 863, pages 694–715, 1994.
17. S. Yovine. A verification tool for real-time systems. *Springer International Journal of Software Tools for Technology Transfer*, 1, 1997.

Iterative Arrays with Small Time Bounds

Thomas Buchholz, Andreas Klein, and Martin Kutrib

Institute of Informatics, University of Giessen
Arndtstr. 2, D-35392 Giessen, Germany
`kutrib@informatik.uni-giessen.de`

Abstract. An iterative array is a line of interconnected interacting finite automata. One distinguished automaton, the communication cell, is connected to the outside world and fetches the input serially symbol by symbol. Sometimes in the literature this model is referred to as cellular automaton with sequential input mode. We investigate deterministic iterative arrays (IA) with small time bounds between real-time and linear-time. It is shown that there exists an infinite dense hierarchy of strictly included complexity classes in that range. The result closes the last gap in the time hierarchy of IAs.

Keywords: Iterative arrays, cellular automata, computational complexity, time hierarchies.

1 Introduction

Devices of interconnected parallel acting automata have extensively been investigated from a computational complexity point of view. The specification of such a system includes the type and specification of the identical automata, their interconnection scheme (which can imply a dimension to the system), a local and/or global transition function and the input and output modes. One-dimensional devices with nearest neighbor connections whose cells are deterministic finite automata are commonly called iterative arrays (IA) if the input mode is sequential to a distinguished communication cell.

Especially for practical reasons and for the design of systolic algorithms a sequential input mode is more natural than the parallel input mode of so-called cellular automata. Various other types of acceptors have been investigated under this aspect (e.g. the iterative tree acceptors in [8]).

In connection with formal language recognition IAs have been introduced in [7] where it was shown that the language family accepted by real-time IAs forms a Boolean algebra not closed under concatenation and reversal. In [6] it is shown that for every context-free grammar a 2-dimensional linear-time IA parser exists.

In [9] a real-time acceptor for prime numbers has been constructed. Pattern manipulation is the main aspect in [1]. A characterization of various types of IAs by restricted Turing machines and several results especially speed-up theorems are given in [10,11,12].

Various generalizations of IAs have been considered. In [15] IAs are studied in which all the finite automata are additionally connected to the communication

cell. Several more results concerning formal languages can be found (e.g. in [16, 17, 18]).

In the field of computational complexity there is a particular interest in infinite hierarchies of complexity classes defined by bounding some resources. In order to obtain dense hierarchies one has to show that only a slight increase in the growth rate of the bounding function yields a new complexity class.

Recently in [13] a time hierarchy for IAs has been proved. For time-computable functions t' and t such that $t' \in o(t)$ a strict inclusion between the corresponding complexity classes is shown. Since an IA can be sped-up from $(n + r(n))$ -time to $(n + \varepsilon \cdot r(n))$ -time for all $\varepsilon > 0$ but real-time is strictly weaker than linear-time [7] this hierarchy is dense in the range above linear-time. Here we close the gap between real-time and linear-time. We show that there exists an infinite dense time hierarchy in between.

Relating our main result for example to Turing machines (TM) it is known that for deterministic one-tape TMs as well as for almost all nondeterministic TMs real-time is as powerful as linear-time. For deterministic real-time machines infinite hierarchies depending on the number of tapes or on the number of heads have been shown. Moreover, for multi- or k -tape TMs ($k \geq 2$) linear-time is known to yield strictly more powerful acceptors than real-time. For any time complexity $n + r(n)$ at most a speed-up to $n + (\varepsilon \cdot r(n))$ ($\varepsilon > 0$), is possible (e.g. [19] for TM results). The remaining gap between real-time and linear-time was filled with an infinite dense hierarchy in [5].

The basic notions and the model in question are defined in the next section. Section 3 is devoted to the hierarchy theorem and some examples. In Section 4 the proof of the Theorem is presented with the help of two lemmas and three tasks for the construction of a corresponding IA.

2 Preliminaries

We denote the rational numbers by $\mathbb{Q}$, the integers by $\mathbb{Z}$, the positive integers $\{1, 2, \dots\}$ by $\mathbb{N}$ and the set $\mathbb{N} \cup \{0\}$ by $\mathbb{N}_0$. The empty word is denoted by ε and the reversal of a word w by w^R . We use $\subseteq$ for inclusions and $\subset$ if the inclusion is strict. For a function $f : \mathbb{N}_0 \rightarrow \mathbb{N}$ we denote its i -fold composition by $f^{[i]}$, $i \in \mathbb{N}$, and define the set of mappings that grow strictly less than f by $o(f) = \{g : \mathbb{N}_0 \rightarrow \mathbb{N} \mid \lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0\}$. If f is strictly increasing then its inverse is defined according to $f^{-1}(n) = \min\{m \in \mathbb{N} \mid f(m) \geq n\}$. The identity function $n \mapsto n$ is denoted by id .

An iterative array is a bi-infinite linear array of finite automata, sometimes called cells, where each of them is connected to its both nearest neighbors (one to the right and one to the left). For convenience we identify the cells by integers. Initially they are in the so-called quiescent state. The input is supplied sequentially to the distinguished communication cell at the origin. For this reason we have two local transition functions. The state transition of all cells but the communication cell depends on the current state of the cell itself and the current states of its both neighbors. The state transition of the communication

cell additionally depends on the current input symbol (or if the whole input has been consumed on a special end-of-input symbol).

The finite automata work synchronously at discrete time steps. More formally:

Definition 1. *An iterative array (IA) is a system $(S, \delta, \delta_0, s_0, \#, A, F)$, where*

1. S is the finite, nonempty set of cell states,
2. A is the finite, nonempty set of input symbols,
3. $F \subseteq S$ is the set of accepting states,
4. $s_0 \in S$ is the quiescent state,
5. $\# \notin A$ is the end-of-input symbol,
6. $\delta : S^3 \rightarrow S$ is the local transition function for non-communication cells satisfying $\delta(s_0, s_0, s_0) = s_0$,
7. $\delta_0 : S^3 \times (A \cup \{\#\}) \rightarrow S$ is the local transition function for the communication cell.

Let $\mathcal{M}$ be an IA. A configuration of $\mathcal{M}$ at some time $t \geq 0$ is a description of its global state which is a pair (w_t, c_t) where $w_t \in A^*$ is the remaining input sequence and $c_t : \mathbb{Z} \rightarrow S$ is a mapping that maps the single cells to their current states. The configuration (w_0, c_0) at time 0 is defined by the input word w_0 and the mapping $c_0(i) = s_0$, $i \in \mathbb{Z}$, while subsequent configurations are chosen according to the global transition function Δ : Let (w_t, c_t) , $t \geq 0$, be a configuration then its successor configuration (w_{t+1}, c_{t+1}) is as follows:

$$\begin{aligned} (w_{t+1}, c_{t+1}) &= \Delta((w_t, c_t)) \iff \\ c_{t+1}(i) &= \delta(c_t(i-1), c_t(i), c_t(i+1)), i \in \mathbb{Z} \setminus \{0\}, \\ c_{t+1}(0) &= \delta_0(c_t(-1), c_t(0), c_t(1), a) \end{aligned}$$

where $a = \#$, $w_{t+1} = \varepsilon$ if $w_t = \varepsilon$, and $a = a_1$, $w_{t+1} = a_2 \cdots a_n$ if $w_t = a_1 \cdots a_n$. Thus, the global transition function Δ is induced by δ and δ_0 .

If the state set is a Cartesian product of some smaller sets $S = S_1 \times S_2 \times \cdots \times S_k$ we will use the notion *register* for the single parts of a state. The concatenation of one of the registers of all cells respectively forms a *track*. A word w is accepted by an IA if at some time i during its course of computation on input w the communication cell becomes accepting.

Definition 2. *Let $\mathcal{M} = (S, \delta, \delta_0, s_0, \#, A, F)$ be an IA.*

1. *A word $w \in A^*$ is accepted by $\mathcal{M}$ if there exists a time step $i \in \mathbb{N}$ such that $c_i(0) \in F$ for $(w_i, c_i) = \Delta^{[i]}((w, c_0))$.*
2. *$L(\mathcal{M}) = \{w \in A^* \mid w \text{ is accepted by } \mathcal{M}\}$ is the language accepted by $\mathcal{M}$.*
3. *Let $t : \mathbb{N}_0 \rightarrow \mathbb{N}$, $t(n) \geq n + 1$, be a mapping and i_w be the minimal time step at which $\mathcal{M}$ accepts a $w \in L(\mathcal{M})$. If all $w \in L(\mathcal{M})$ are accepted within $i_w \leq t(|w|)$ time steps, then L is said to be of time complexity t .*

The family of all languages which can be accepted by an IA with time complexity t is denoted by $\mathcal{L}_t(\text{IA})$. If t equals the function $n + 1$ acceptance is said to be in *real-time* and we write $\mathcal{L}_{rt}(\text{IA})$. The *linear-time* languages $\mathcal{L}_{lt}(\text{IA})$ are defined according to $\mathcal{L}_{lt}(\text{IA}) = \bigcup_{k \in \mathbb{Q}, k \geq 1} \mathcal{L}_{k \cdot n}(\text{IA})$.

In order to prove infinite dense time hierarchies in almost all cases honest time bounding functions are required. Usually the notion “honest” is concretized in terms of the computability or constructibility of the function with respect to the device in question. Here we will use time-constructible functions in IAs.

Definition 3.

1. A strictly increasing function $f : \mathbb{N} \rightarrow \mathbb{N}$ is IA-time-constructible iff there exists an IA $(S, \delta, \delta_0, s_0, \#, A, F)$ such that for all $i \in \mathbb{N}$

$$c_i(0) \in F \iff \exists n \in \mathbb{N} : i = f(n)$$

where $(\varepsilon, c_i) = \Delta^{[i]}((\varepsilon, c_0))$.

2. The set of all IA-time constructible functions is denoted by $\mathcal{F}(\text{IA})$.
3. The set of their inverses is $\mathcal{F}^{-1}(\text{IA}) = \{f^{-1} \mid f \in \mathcal{F}(\text{IA})\}$.

Thus, an iterative array that time constructs a function f becomes accepting on an empty input exactly at the time steps $f(1), f(2), \dots$. Note that since an $f \in \mathcal{F}(\text{IA})$ is strictly increasing it holds $f^{-1}(n) \leq n$ for all $n \geq 1$.

The family $\mathcal{F}(\text{IA})$ is very rich. It includes $id!$, k^{id} , id^k , $id + \lfloor \sqrt{id} \rfloor$, $id + \lfloor \log \rfloor$, etc., where $k \geq 1$ is an integer. It is closed under the operations addition of constants, addition, iterated addition, multiplication, composition, minimum, maximum etc. [14]. In [9] the function $n \mapsto p_n$ where p_n denotes the n th prime number has shown to be IA-time constructible. In [13] the time-constructibility is related to the time-computability in Turing machines. Further results can be found in [3,4].

3 The Hierarchy between Real-Time and Linear-Time

The hierarchy is proved by a specific witness language which is defined dependent on a mapping $h : \mathbb{N} \rightarrow \mathbb{N}$. Later on h is related to the time complexity in question.

$$\begin{aligned} L_h = \{ \$^{h(m)-(m+1)^2+1} w_1 \$ w_2 \$ \dots \$ w_m \$ y \mid m \in \mathbb{N} \\ \quad \wedge y, w_i \in \{0, 1\}^m, 1 \leq i \leq m \\ \quad \wedge \exists 1 \leq j \leq m : y = w_j \} \end{aligned}$$

The following theorem is the main result of the paper.

Theorem 4. Let $r : \mathbb{N}_0 \rightarrow \mathbb{N}$ and $r' : \mathbb{N}_0 \rightarrow \mathbb{N}$ be two functions. If $r \in \mathcal{F}^{-1}(\text{IA})$ and $r' \in o(r)$ then

$$\mathcal{L}_{id+r'}(\text{IA}) \subset \mathcal{L}_{id+r}(\text{IA})$$

The theorem is proved by Lemma 6 and Lemma 7 where it is shown that a specific instance of L_h is not acceptable with time complexity $id + r'$ but is acceptable with time complexity $id + r$. The inclusion $\mathcal{L}_{id+r'}(\text{IA}) \subseteq \mathcal{L}_{id+r}(\text{IA})$ follows immediately from the definitions.

The following example for hierarchies are based on natural functions.

Example 5. Since $\mathcal{F}(\text{IA})$ is closed under composition and contains 2^{id} and id^k , $k \geq 1$, the functions $\log^{[i]}$, $i \geq 1$, and $\sqrt[k]{id}$ are belonging to $\mathcal{F}^{-1}(\text{IA})$.¹ Therefore, an application to the hierarchy theorem yields

$$\mathcal{L}_{rt}(\text{IA}) \subset \cdots \subset \mathcal{L}_{id+\log^{[i+1]}}(\text{IA}) \subset \mathcal{L}_{id+\log^{[i]}}(\text{IA}) \subset \cdots \subset \mathcal{L}_{lt}(\text{IA})$$

and

$$\mathcal{L}_{rt}(\text{IA}) \subset \cdots \subset \mathcal{L}_{id+id^{\frac{1}{i+1}}}(\text{IA}) \subset \mathcal{L}_{id+id^{\frac{1}{i}}}(\text{IA}) \subset \cdots \subset \mathcal{L}_{lt}(\text{IA})$$

or in combinations e.g.,

$$\begin{aligned} \mathcal{L}_{rt}(\text{IA}) \subset \cdots \subset \mathcal{L}_{id+(\log^{[j+1]})^{\frac{1}{i+1}}}(\text{IA}) \subset \mathcal{L}_{id+(\log^{[j+1]})^{\frac{1}{i}}}(\text{IA}) \subset \cdots \\ \cdots \subset \mathcal{L}_{id+(\log^{[j]})^{\frac{1}{i+1}}}(\text{IA}) \subset \mathcal{L}_{id+(\log^{[j]})^{\frac{1}{i}}}(\text{IA}) \subset \cdots \subset \mathcal{L}_{lt}(\text{IA}) \end{aligned}$$

4 Proof of the Hierarchy

The object of the present section is to prove the lemmas from which the hierarchy follows. At first we show that a specific instance of the language L_h cannot be accepted in $(id + r')$ -time by any deterministic IA. Therefore let r be a function in $\mathcal{F}^{-1}(\text{IA})$. Then there exists a function $f_r \in \mathcal{F}(\text{IA})$ such that $r = f_r^{-1}$.

Now let $h_r : \mathbb{N}_0 \rightarrow \mathbb{N}$ be defined as

$$h_r(n) = f_r((n+1)^2)$$

Lemma 6. *Let $r : \mathbb{N}_0 \rightarrow \mathbb{N}$ and $r' : \mathbb{N}_0 \rightarrow \mathbb{N}$ be two functions. If $r \in \mathcal{F}^{-1}(\text{IA})$ and $r' \in o(r)$ then*

$$L_{h_r} \notin \mathcal{L}_{id+r'}(\text{IA})$$

Proof. Contrarily, assume $L_{h_r} = \{\$^{h_r(m)-(m+1)^2+1}w_1\$ \cdots \$w_m\$y \mid m \in \mathbb{N} \wedge y, w_i \in \{0, 1\}^m, 1 \leq i \leq m \wedge \exists 1 \leq j \leq m : y = w_j\}$ is acceptable by $\mathcal{M} = (S, \delta, \delta_0, s_0, \#, A, F)$ with time complexity $id + r'$.

Now we consider the situation at time $n - m$ where n denotes the length of the input and $m < n$. The remaining computation of the communication cell depends on the last m input symbols and the states of the cells $-m -$

¹ Actually, the inverses of 2^{id} and id^k are $\lceil \log \rceil$ and $\lceil id^{\frac{1}{k}} \rceil$ respectively but for convenience we simplify the notation.

$r'(n), \dots, 0, \dots, m + r'(n)$ only. Due to their distance from the communication cell all the other cells cannot influence the overall computation result.

So we are concerned with at most $|S|^{2 \cdot (m+r'(n))+1}$ distinct situations. Related to L_{h_r} where m is the length of the subword y we obtain

$$|S|^{2 \cdot (m+r'(h_r(m)))+1} \leq (|S|^3)^{m+r'(h_r(m))}.$$

Since $r = f_r^{-1}$ and $h_r(n) = f_r((n+1)^2)$ we have

$$r(h_r(m)) = f_r^{-1}(f_r((m+1)^2)) = (m+1)^2.$$

Moreover, from $r' \in o(r)$ it follows $r'(h_r(m)) \in o(m^2)$.

Now choose $k \in \mathbb{N}$ such that $2^{\frac{k}{8}} \geq |S|^3$. For all sufficiently large m it holds $r'(h_r(m)) < \frac{m^2}{k}$ and further

$$(|S|^3)^{m+r'(h_r(m))} \leq (|S|^3)^{m+\frac{m^2}{k}} \leq (|S|^3)^{\frac{2 \cdot m^2}{k}} \leq 2^{\frac{m^2}{4}}.$$

On the other hand, for all m -subsets $U = \{w_1, \dots, w_m\}$ of $\{0, 1\}^m$, $m \in \mathbb{N}$, there exists a word w_U such that $y \in U \iff w_U y \in L_{h_r}$. (E.g. $w_U = \$^{h(m)-(m+1)^2+1} w_1 \$ \dots \$ w_m \pounds$). Moreover, for each two distinct m -subsets U and V of $\{0, 1\}^m$ there exists a word $y \in (U \setminus V) \cup (V \setminus U)$. Thus

$$w_U y \in L_{h_r} \iff w_V y \notin L_{h_r}.$$

The number of m -subsets U of $\{0, 1\}^m$ is

$$\binom{2^m}{m} > \left(\frac{2^m - m}{m} \right)^m \geq \left(\frac{2^{\frac{m}{2}}}{m} \right)^m = \left(2^{\frac{m}{2} - \log(m)} \right)^m > 2^{\frac{m^2}{4}}$$

for all sufficiently large m .

Since $\mathcal{M}$ can distinguish at most $2^{\frac{m^2}{4}}$ different situations at time $n - m$ there must exist two distinct m -subsets U and V such that the situation after processing w_U and w_V is identical. We conclude for $y \in (U \setminus V) \cup (V \setminus U)$ that $\mathcal{M}$ accepts $w_U y$ iff it accepts $w_V y$. But exactly one of the words belongs to L_{h_r} , what is a contradiction. $\square$

It remains to show that L_{h_r} is acceptable with time complexity $id + r$. The following results are devoted to the construction of an appropriate IA.

Lemma 7. *Let $r : \mathbb{N}_0 \rightarrow \mathbb{N}$ be a function such that $r \in \mathcal{F}^{-1}(\text{IA})$ then*

$$L_{h_r} \in \mathcal{L}_{id+r}(\text{IA})$$

Proof. In what follows we construct an IA $\mathcal{M}$ that accepts L_{h_r} with time complexity $id + r$. Since $\mathcal{L}_{id+r}(\text{IA})$ is closed under intersection and contains the regular languages we may restrict our considerations to input words of the form $\$^* \{0, 1\}^* \$^* \{0, 1\}^* \$ \dots \$ \{0, 1\}^+ \pounds \{0, 1\}^+$, say $w = \$^p u \pounds y$ where $u = w_1 \$ w_2 \$ \dots \$ w_q$ for $p \geq 0$, $q \geq 1$ and $w_1, \dots, w_q, y \in \{0, 1\}^*$.

$\mathcal{M}$ is designed to perform three tasks in parallel. The first one is to check whether $|w| = h_r(|y|)$, and similarly the second one to ensure that $|u\clubsuit y| = |y|^2 + 2|y|$. Finally, the third task is to verify that $|w_1| = \dots = |w_q| = |y|$ and that $w_j = y$ for some $1 \leq j \leq q$. The input is accepted if and only if all tasks succeed.

In this case if we set $m = |y|$ we have $|u\clubsuit| = qm + q$ and thus

$$qm + q + m = |u\clubsuit y| = m^2 + 2m$$

from which $q = m$ follows. Further it holds

$$p = |w| - |u\clubsuit y| = h_r(m) - m^2 - 2m = h_r(m) - (m + 1)^2 + 1$$

and hence $w \in L_{h_r}$.

Now we consider the time requirements of the three tasks mentioned above what proves the lemma. $\square$

Task 1. First we need to derive two constructibility properties of the functions in question. Since $\mathcal{F}(\text{IA})$ contains the functions f_r , id and id^2 and is closed under addition of constants and under composition, the function $h_r = f_r((id + 1)^2)$ belongs to $\mathcal{F}(\text{IA})$. Moreover, $h' = h_r + 1$ and $g' = h_r - id$ are IA-time-constructible functions which is obvious for h' and can be seen for g' as follows.

Let $m \geq 1$. Since f_r is strictly increasing it holds

$$f_r(m + k) \geq f_r(m) + k \geq m + k$$

for all $k \in \mathbb{N}$. Observe that g' is strictly increasing, since

$$\begin{aligned} g'(m + 1) &= f_r((m + 2)^2) - (m + 1) \geq f_r((m + 1)^2) + m + 2 \\ &> f_r((m + 1)^2) - m = g'(m) \\ &\geq (m + 1)^2 - m = m^2 + m + 1 \\ &> m \end{aligned}$$

Thus, $g' + id (= h_r)$ is IA-time-constructible. The following result has been shown in [2] (in terms of cellular spaces). Let $f \geq id$ be a strictly increasing function, then

$$f + id \in \mathcal{F}(\text{IA}) \implies f \in \mathcal{F}(\text{IA})$$

Therefore, we obtain that g' is IA-time-constructible.

In order to realize the first task $\mathcal{M}$ basically simulates a time constructor for g' and another one for h' on two different tracks. The task succeeds if the time constructor for g' marks the communication cell at the time step it fetches the $\clubsuit$ symbol from the input and the next marking is by the time constructor for h' at the time step it fetches the first end-of-input symbol.

In such a case we have $|w| = h'(m) - 1 = h_r(m)$ for some $m \in \mathbb{N}$. Moreover $|y| = h'(m) - g'(m) - 1 = m$. It remains to show that the constructors do not

interfere: Suppose $\mathcal{M}$ fetches $\pounds$ at time step $g'(m)$ for some $m \in \mathbb{N}$. For all $m' \in \mathbb{N}$ we have

$$\begin{aligned} g'(m') &< h'(m') = f_r((m' + 1)^2) + 1 \\ &< f_r((m' + 1)^2) + m' + 2 \\ &\leq f_r((m' + 2)^2) - m' - 1 = g'(m' + 1) \\ &< h'(m' + 1) \end{aligned}$$

It follows that after being marked by g' the communication cell is next marked by h' , then next by g' , and so on alternating.

Task 2. This task is realized analogously to the previous task. We just have to use time constructors for $id^2 + id$ and $(id + 1)^2$ which are started as soon as the first symbol different from $\$$ appears in the input. It follows $|u\pounds y| = m^2 + 2m$ for some $m \in \mathbb{N}$ and $|y| = (m + 1)^2 - (m^2 + m) - 1 = m$.

Task 3. For the last task an acceptor for $L = \{w_1\$w_2\$ \cdots \$w_q\pounds y \mid q \in \mathbb{N} \wedge \forall 1 \leq i \leq q : w_i, y \in \{0, 1\}^* \wedge |w_1| = \cdots = |w_q| = |y| \wedge \exists 1 \leq j \leq q : w_j = y\}$ is simulated which accepts L with time complexity $2id$ (see Lemma 8). Whereby the simulation is started when the first symbol different from $\$$ is fetched.

Hence the last task requires at most $p + 2|u\$y|$ time steps.

If all tasks succeed w is accepted at time step

$$\begin{aligned} p + 2|u\$y| &= h_r(m) - (m + 1)^2 + 1 + 2(m^2 + 2m) \\ &= h_r(m) + m^2 + 2m < h_r(m) + (m + 1)^2 \\ &= h_r(m) + r(h_r(m)) \\ &= |w| + r(|w|) \end{aligned}$$

It follows $L_{h_r} \in \mathcal{L}_{id+r}(\text{IA})$.

Lemma 8. $L = \{w_1\$w_2\$ \cdots \$w_q\pounds y \mid q \in \mathbb{N} \wedge \forall 1 \leq i \leq q : w_i, y \in \{0, 1\}^* \wedge |w_1| = \cdots = |w_q| = |y| \wedge \exists 1 \leq j \leq q : w_j = y\} \in \mathcal{L}_{2 \cdot id}(\text{IA})$.

Proof. Since $\mathcal{L}_{it}(\text{IA}) = \mathcal{L}_{2 \cdot id}(\text{IA})$ [11] it suffices to construct an acceptor $\mathcal{M}$ for L with time complexity $3 \cdot id$. W.l.o.g. we may assume that the input w is of the form $\{0, 1\}^* \$ \{0, 1\}^* \$ \cdots \$ \{0, 1\}^+ \pounds \{0, 1\}^*$, say $w = u\pounds y$ with $u = w_1\$w_2\$ \cdots \$w_q$ and $w_1, \dots, w_q, y \in \{0, 1\}^*, q \geq 1$.

Basically, $\mathcal{M}$ stores the input symbols into successive cells to the right whereby the i th fetched symbol is piped through the cells such that it is stored into cell $i - 1$. Additionally, during the pipe process y is compared with w_i , $1 \leq i \leq q$. The result of the comparison can verify that $|y| = |w_i|$ for all $1 \leq i \leq q$ and that at least one w_i matches y .

The piped symbol $\pounds$ plays a distinguished role for initiating the comparison. The cells storing $\$$ are used to remember partial comparison results. If the first end-of-input symbol passes through the cell storing the symbol $\pounds$ a leftward moving signal is generated which collects the partial comparison results.

More precisely, the cells of $\mathcal{M}$ are equipped with a pipe and a store register which are initially empty. If a cell detects that the pipe register of its left neighbor cell is filled then it takes over the content and stores it into its own store register if it is empty and into its own pipe register otherwise. If a cell is passed through by the symbol $\clubsuit$ its subsequent behavior depends on the content of its store register as follows.

If its store register contains 0 or 1 it waits for the first unmarked symbol 0 or 1 in the pipe register of its left neighbor. If this symbol is equal to the content of its own store register then it marks its both register contents by $+$ and otherwise by $-$. In any case it takes over the symbol into its own pipe register.

If its store register contains $\$$ it waits for the first end-of-input symbol in the pipe register of its left neighbor. Meanwhile it behaves as follows: If it detects an unmarked symbol 0 or 1 (of y) in the pipe register of its left neighbor then it marks its own store register content by $-$. This corresponds to the situation that at least one symbol of y could not be compared to a symbol in the subword w_i which proceeds the occurrence of that $\$$ in w , i.e., $|y| > |w_i|$. Additionally, it copies the symbol in the pipe register of its left neighbor into its own pipe register, whereby existing marks are removed. Thus subsequently the symbols of y can be compared to the symbols of the next subword w_{i+1} .

Further, after the first end-of-input symbol has reached the cell which has stored the symbol $\clubsuit$ a signal is generated which moves leftward with maximal speed. Each cell which is reached by this signal is turned into an accepting state iff none of the cells already passed through contains an marked (by $-$) symbol $\$$ or an unmarked symbol 0 or 1 in its store register and, additionally, at least one sequence of cells (between two consecutive $\$$ -cells) has been passed through in which all store registers are marked by $+$.

Thus the communication cell of $\mathcal{M}$ becomes accepting iff $|w_i| \leq |y|$ and $|w_i| \geq |y|$ for all $1 \leq i \leq q$ and there is at least one word w_j which matches y (i.e. $w \in L$).

Moreover it takes $2|u| + 1$ time steps to move the symbol $\clubsuit$ into the store register of cell $|u|$. Since the signal reaches the communication cell after at most $|u| + 1$ additional time steps, in total w is accepted in

$$2|u| + 1 + |u| + 1 < 3|u\clubsuit y| = 3|w|$$

time steps. Hence, L belongs to $\mathcal{L}_{3.id}(\text{IA})$. □

References

1. Beyer, W. T. *Recognition of topological invariants by iterative arrays*. Technical Report TR-66, MIT, Cambridge, Proj. MAC, 1969.
2. Buchholz, Th. and Kutrib, M. *On the power of one-way bounded cellular time computers*. Developments in Language Theory, 1997, pp. 365–375.
3. Buchholz, Th. and Kutrib, M. *Some relations between massively parallel arrays*. Parallel Comput. 23 (1997), 1643–1662.
4. Buchholz, Th. and Kutrib, M. *On time computability of functions in one-way cellular automata*. Acta Inf. 35 (1998), 329–352.

5. Buchholz, Th., Klein, A., and Kutrib, M. *Deterministic turing machines in the range between real-time and linear-time*. To appear.
6. Chang, J. H., Ibarra, O. H., and Palis, M. A. *Parallel parsing on a one-way array of finite-state machines*. IEEE Trans. Comput. C-36 (1987), 64–75.
7. Cole, S. N. *Real-time computation by n -dimensional iterative arrays of finite-state machines*. IEEE Trans. Comput. C-18 (1969), 349–365.
8. Čulik II, K. and Yu, S. *Iterative tree automata*. Theoret. Comput. Sci. 32 (1984), 227–247.
9. Fischer, P. C. *Generation of primes by a one-dimensional real-time iterative array*. J. Assoc. Comput. Mach. 12 (1965), 388–394.
10. Ibarra, O. H. and Jiang, T. *On one-way cellular arrays*. SIAM J. Comput. 16 (1987), 1135–1154.
11. Ibarra, O. H. and Palis, M. A. *Some results concerning linear iterative (systolic) arrays*. J. Parallel and Distributed Comput. 2 (1985), 182–218.
12. Ibarra, O. H. and Palis, M. A. *Two-dimensional iterative arrays: Characterizations and applications*. Theoret. Comput. Sci. 57 (1988), 47–86.
13. Iwamoto, C., Hatsuyama, T., Morita, K., and Imai, K. *On time-constructible functions in one-dimensional cellular automata*. Fundamentals of Computation Theory 1999, LNCS 1684, 1999, pp. 317–326.
14. Mazoyer, J. and Terrier, V. *Signals in one dimensional cellular automata*. Theoret. Comput. Sci. 217 (1999), 53–80.
15. Seiferas, J. I. *Iterative arrays with direct central control*. Acta Inf. 8 (1977), 177–192.
16. Seiferas, J. I. *Linear-time computation by nondeterministic multidimensional iterative arrays*. SIAM J. Comput. 6 (1977), 487–504.
17. Smith III, A. R. *Real-time language recognition by one-dimensional cellular automata*. J. Comput. System Sci. 6 (1972), 233–253.
18. Terrier, V. *On real time one-way cellular array*. Theoret. Comput. Sci. 141 (1995), 331–335.
19. Wagner, K. and Wechsung, G. *Computational Complexity*. Reidel Publishing, Dordrecht, 1986.

Embedding Fibonacci Cubes into Hypercubes with $\Omega(2^{cn})$ Faulty Nodes ^{*}

Rostislav Caha and Petr Gregor

Charles University, Dep. of Theoretical Computer Science,
Malostranské nám. 25, 118 00 Prague 1, Czech Republic.
{caha,gregor}@kti.mff.cuni.cz

Abstract. Fibonacci Cubes are special subgraphs of hypercubes based on Fibonacci numbers. We present a construction of a direct embedding of a Fibonacci Cube of dimension n into a faulty hypercube of dimension n with less or equal $2^{\lceil \frac{n}{4} \rceil - 1}$ faults. In fact, there exists a direct embedding of a Fibonacci Cube of dimension n into a faulty hypercube of dimension n with at most $\frac{2^n}{4f_n}$ faults (f_n is the n -th Fibonacci number). Thus the number $\phi(n)$ of tolerable faults grows exponentially with respect to dimension n , $\phi(n) = \Omega(2^{cn})$, for $c = 2 - \log_2(1 + \sqrt{5}) \doteq 0.31$. On the other hand, $\phi(n) = O(2^{dn})$, for $d = (8 - 3 \log_2 3)/4 \doteq 0.82$. As a corollary, there exists a nearly polynomial algorithm constructing a direct embedding of a Fibonacci Cube of dimension n into a faulty hypercube of dimension n (if it exists) provided that faults are given on input by enumeration. However, the problem is NP-complete, if faults are given on input with an asterisk convention.

1 Introduction

The hypercube network (the n -cube or the Boolean cube) is one of the most popular parallel architectures. It has been shown that many other parallel networks can be efficiently emulated by hypercube ([12]). Its nice properties follow from a regular recursive structure and a rich interconnection topology. Recently, its robustness has been studied ([8], [14]) and fault-tolerant embeddings have been found ([2], [13]).

Fibonacci Cubes proposed in [10] are special subgraphs of hypercubes based on Fibonacci numbers. They are much sparser than hypercubes, the number of edges of the Fibonacci Cube of dimension n is $(2(n+1)f_{n+2} - (n+2)f_{n+1})/5$ and the number of vertices of the Fibonacci Cube of dimension n is f_{n+2} where f_n is the n -th Fibonacci number ([10]). Thus the number of vertices of Fibonacci Cubes increases more slowly than the number of vertices of hypercubes but as well as hypercubes, Fibonacci Cubes have a self-similar recursive structure useful for the design of parallel and distributed algorithms ([1], [4], [5]). Various modifications of Fibonacci Cubes have been proposed in [3], [15].

^{*} This work was supported by the Grant Agency of the Czech Republic under the grant no. 201/98/1451.

In this paper, we consider the following problem: How many nodes in a hypercube of dimension n can be faulty under condition that a Fibonacci Cube of dimension n is isomorphic to a subgraph of the hypercube? The question was raised in [10] (a modification in [15]), but no bound was given. In [9], a direct embedding of $\lfloor \frac{n}{2} \rfloor$ -th order Generalized Fibonacci Cube of dimension $n + \lfloor \frac{n}{2} \rfloor$ into a hypercube of dimension n with no more than three faults was given.

We present an effective construction of a direct embedding of a Fibonacci Cube of dimension n into a faulty hypercube of dimension n with less or equal $2^{\lceil \frac{n}{4} \rceil - 1}$ faults and we prove that there exists a direct embedding of a Fibonacci Cube of dimension n into a faulty hypercube of dimension n with at most $\frac{2^n}{4f_n}$ faults. Thus the number $\phi(n)$ of tolerable faults grows exponentially with respect to dimension n , $\phi(n) = \Omega(2^{cn})$, for $c = 2 - \log_2(1 + \sqrt{5}) \doteq 0.31$.

We also give an upper bound $\phi(n) = O(2^{dn})$, for $d = 2 - \frac{3}{4} \log_2 3 \doteq 0.82$.

As a corollary, there exists a nearly polynomial algorithm deciding whether the existence of a direct embedding (and in the affirmative case constructing one) of a Fibonacci Cube of dimension n into a faulty hypercube of dimension n provided that faults are given on input by enumeration. Last but not least, we inquire into an embedding problem under circumstances that the faulty nodes are given on input with an asterisk convention. We show that this problem is NP-complete.

2 Preliminaries and Notations

The *hypercube* Q_n of dimension n is the network with 2^n nodes labeled with binary strings of length n with an edge between two nodes whenever their labels differ exactly by one bit. A *direct embedding* of a graph $G = (V, E)$ into a graph $H = (V', E')$ is an injective mapping $g : V \rightarrow V'$ preserving edges: $\{u, v\} \in E \Rightarrow \{g(u), g(v)\} \in E'$. Thus G is isomorphic to a subgraph of H .

We assume that there are only faulty nodes in Q_n and no faulty link. A *faulty hypercube* with a set of faults F (we write $Q_n \setminus F$) is the induced subgraph of $Q_n = (V, E)$ on the set $V \setminus F$.

The original definition of a Fibonacci Cube ([10]) was based on *Zeckendorf's theorem* [6] and *Fibonacci codes*. We use an equivalent recursive definition.

Definition 1. For $i = 0, \dots, n$ define sets V_i of binary strings of length i

$$\begin{aligned} V_0 &= \{\lambda\}, & V_1 &= \{0, 1\}, \\ V_i &= \{0\alpha \mid \alpha \in V_{i-1}\} \cup \{10\alpha \mid \alpha \in V_{i-2}\}, & \text{for } i > 1. \end{aligned}$$

A *Fibonacci Cube* FC_n of dimension n is the induced subgraph of Q_n on the set V_n . For a simplicity, let $v \in FC_n$ (resp. $v \in Q_n$) denote that v is a vertex of the Fibonacci Cube (resp. the hypercube) of dimension n .

We give a characterization of nodes from Fibonacci Cubes, see [11].

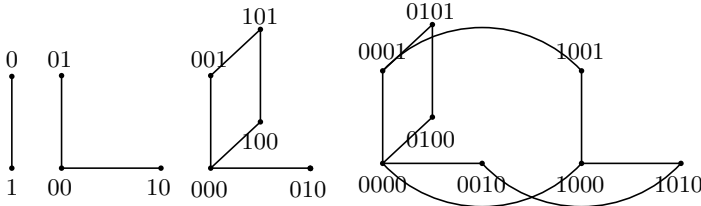


Fig. 1. FC_1, FC_2, FC_3, FC_4

Lemma 1. *Let $v = v_n \dots v_1$ be a binary string. Then*

$$v \in FC_n \quad \text{iff} \quad v_i = 1 \Rightarrow v_{i-1} = 0 \text{ for all } i = n, \dots, 2.$$

Finally, we introduce some definitions concerning the group $\text{Aut}(Q_n)$ of all automorphisms of Q_n .

Definition 2. *For a permutation π of $\{1, \dots, n\}$ and a subset $A \subseteq \{1, \dots, n\}$, let f_π and h_A denote the bijection of the hypercube Q_n such that*

$$f_\pi(v_n \dots v_1) = (v_{\pi(n)} \dots v_{\pi(1)}),$$

$$h_A(v_n \dots v_1) = (w_n \dots w_1), \quad \text{where } w_i = \begin{cases} v_i & \text{if } i \notin A, \\ \bar{v}_i & \text{if } i \in A. \end{cases}$$

Then f_π and h_A are automorphisms of Q_n and set

$$\begin{aligned} R(Q_n) &= \{f_\pi \mid \pi \text{ is a permutation of } \{1, \dots, n\}\}, \\ S(Q_n) &= \{h_A \mid A \subseteq \{1, \dots, n\}\}. \end{aligned}$$

The following lemma is a well known fact ([12]).

Lemma 2. *$R(Q_n)$ and $S(Q_n)$ are subgroups of $\text{Aut}(Q_n)$ and for every automorphism $g \in \text{Aut}(Q_n)$ there exist exactly one $f_\pi \in R(Q_n)$ and $h_A \in S(Q_n)$ with $g = f_\pi \circ h_A$.*

3 Characterization of Embeddings

First we characterize all direct embeddings of FC_n into $Q_n \setminus F$. The following lemma states that any direct embedding can be uniquely extended to an automorphism of Q_n .

Lemma 3. *Let g be a direct embedding of FC_n into $Q_n \setminus F$. There exist exactly one $f_\pi \in R(Q_n)$ and $h_A \in S(Q_n)$ such that g is the domain restriction of hypercube automorphism $g' = f_\pi \circ h_A$ on FC_n .*

Proof. Omitted for full version.

In subsequent section we suggest an algorithm that for a suitable set of faults $F \subseteq Q_n$ finds a direct embedding of the Fibonacci Cube of dimension n into the faulty hypercube of dimension n with faults F . By Lemma 3, any embedding $g : FC_n \rightarrow Q_n \setminus F$ can be extended to an automorphism of Q_n and thus can be composed of a negation and a permutation of bits in node labels. By Lemma 1, a vertex of Q_n belongs to FC_n iff it has not two consecutive ones. The key idea is to find a set A and a permutation π of $\{1, \dots, n\}$ such that $g = f_\pi \circ h_A$ maps all faults to vertices with two consecutive ones. Then the domain restriction of g^{-1} on FC_n is the required embedding.

Before the algorithm, we define an optimal number of tolerable faults.

Definition 3. For an integer $n \geq 1$, let $\phi(n)$ denote the greatest integer such that for every set F of vertices of Q_n with $|F| \leq \phi(n)$ there exists a direct embedding g of FC_n into $Q_n \setminus F$.

Fig. 2 was obtained by inspection of all cases and verified by computer ([7]).

n	1	2	3	4	5	6
$\phi(n)$	0	1	1	3	5	≥ 7

Fig. 2. Values of $\phi(n)$ for a small dimension

4 Embedding Algorithm

First observe that $|FC_4| = f_6 = 8 = \frac{2^4}{2}$. Let $\overline{FC_4}$ denote the complement of FC_4 in Q_4 . Consider the automorphism $g_0 = f_\pi \circ h_{\{1,2,3,4\}}$ where $\pi(1, 2, 3, 4) = (1, 3, 2, 4)$, then $g_0(FC_4) = \overline{FC_4}$. This observation is exploited in the following algorithm.

Assume that a faulty hypercube Q_n with a set F of faults is given with $n > 5$. Then there exists a graph $G = (V, E)$ isomorphic to Q_{n-4} such that $Q_n = Q_4 \times G$. Set $F_1 = F \cap (FC_4 \times G)$, $F_2 = F \cap (\overline{FC_4} \times G)$ then $F = F_1 \cup F_2$ and $F_1 \cap F_2 = \emptyset$. Thus $|F_1| \geq \frac{|F|}{2}$ or $|F_2| \geq \frac{|F|}{2}$. For every automorphism g' of G define an automorphism g of Q_n such that (id is the identical automorphism)

$$g = \begin{cases} id \times g' & \text{if } |F_2| \geq \frac{|F|}{2}, \\ g_0 \times g' & \text{if } |F_1| > \frac{|F|}{2}. \end{cases}$$

Then g is an automorphism of Q_n such that if $|F_2| \geq \frac{|F|}{2}$ then $g(F_2) \cap FC_n = \emptyset$, if $|F_1| > \frac{|F|}{2}$ then $g(F_1) \cap FC_n = \emptyset$. Therefore if g' maps subcubes containing the less set from F_1 and F_2 into the complement of FC_n we obtain that $g(F) \cap FC_n = \emptyset$. To construct g' we apply this idea recursively and for dimension $n \leq 6$ we search for g' exhaustively.

Algorithm 1

1. Initialize $A = \emptyset$, $i = n$, let π be the empty partial permutation.
2. Repeat this step until $i \leq 6$: Let

$$F_1 = \{v \in F \mid (v_i, \dots, v_{i-3}) \text{ does not contain two consecutive ones}\},$$

$$F_2 = \{v \in F \mid (v_i, \dots, v_{i-3}) \text{ contains two consecutive ones}\}.$$

- If $|F_2| \geq \frac{|F|}{2}$ then set $A \leftarrow A$, $\pi(j) \leftarrow j$ for $j \in \{i, i-1, i-2, i-3\}$ and $F \leftarrow F_1$.
 If $|F_1| > \frac{|F|}{2}$ then set $A \leftarrow A \cup \{i, i-1, i-2, i-3\}$, $\pi(i) \leftarrow i$, $\pi(i-1) \leftarrow i-2$, $\pi(i-2) \leftarrow i-1$, $\pi(i-3) \leftarrow i-3$ and $F \leftarrow F_2$. Set $i \leftarrow i-4$.
 3. If $F \neq \emptyset$ and $i \leq 6$ then set $F' = \{(v_i, \dots, v_1) \mid (v_n, \dots, v_1) \in F\}$. By the exhaustive search we find a permutation $g' = f_{\pi'} \circ h_{A'}$ of Q_i (if it exists) such that $g'(F') \cap FC_i = \emptyset$. Set $A \leftarrow A \cup A'$, $\pi(j) \leftarrow \pi'(j)$ for $j \in \{i, i-1, \dots, 1\}$.

To analyze Algorithm 1, we introduce the following function.

Definition 4. If $n \leq 5$ then $p(n) = \phi(n)$, $p(6) = 7$ and for $n > 6$

$$p(n) = \begin{cases} 3 \cdot 2^{\frac{n-4}{4}} & \text{if } n \equiv 0 \pmod{4}, \\ 5 \cdot 2^{\frac{n-5}{4}} & \text{if } n \equiv 1 \pmod{4}, \\ 7 \cdot 2^{\frac{n-6}{4}} & \text{if } n \equiv 2 \pmod{4}, \\ 2^{\frac{n-3}{4}} & \text{if } n \equiv 3 \pmod{4}. \end{cases}$$

Lemma 4. Let p be the function from Definition 4. Then

$$p(n) \geq 2^{\lceil \frac{n}{4} \rceil - 1}, \quad \text{for all } n > 1. \quad (1)$$

Proof. Clearly, $p(n) = 2p(n-4)$ and by recurrency we obtain (1) from Fig. 2.

Lemma 5. Let $n \geq 1$ and $F \subseteq Q_n$ be a set of faults. If $|F| \leq p(n)$ then Algorithm 1 outputs a set A and a permutation π such that

$$g(F) \cap FC_n = \emptyset, \quad \text{for } g = (f_\pi \circ h_A).$$

Proof. Let k' be the number of repetitions of Step 2 and let F_k denote the set F after the k -th repetition (and $F = F_0$ is the original set of faults). Thus $F = \bigcup_{k=0}^{k'} (F \setminus F_k) \cup F_{k'}$. By the observation before Algorithm 1, we deduce that $g(F \setminus F_k) \cap FC_n = \emptyset$, for all k . If $|F| \leq p(n)$ then $|F'| \leq \phi(i)$ in Step 3 since $|F_k| < \frac{|F_{k-1}|}{2}$. Thus Algorithm 1 succesively finds g' and $g(F_{k'}) \cap FC_n = \emptyset$. $\square$

Remark 1. By $g(F) \cap FC_n = \emptyset$, we deduce that $g^{-1} : FC_n \rightarrow Q_n \setminus F$ is a direct embedding of the Fibonacci Cube FC_n into the faulty hypercube $Q_n \setminus F$ and $\phi(n) \geq p(n)$.

Now, we will prove a stronger estimate of the function ϕ than $\phi(n) \geq p(n)$.

Theorem 1. *For every $n \geq 6$*

$$\phi(n) \geq \frac{2^n}{4f_n} \geq \frac{\sqrt{5}}{4} 2^{n(2-\log_2(1+\sqrt{5}))} = \Omega(2^{0.31n}).$$

Proof. The automorphism group $Aut(Q_n)$ is transitive and for $u, v \in Q_n$ the number of automorphisms g of $Aut(Q_n)$ with $g(u) = v$ is $n!$. Thus for $A, B \subseteq Q_n$ we obtain

$$\sum_{g \in Aut(Q_n)} |g(A) \cap B| = \sum_{u \in A, v \in B} n! = |A| \cdot |B| \cdot n!$$

and hence we deduce that there exists an automorphism $g \in Aut(Q_n)$ with $|g(A) \cap B| \leq \frac{|A| \cdot |B|}{2^n}$ because $|Aut(Q_n)| = n!2^n$.

Assume that F is the set of faults with $|F| \leq \frac{2^n}{4f_n}$. Set $F' = \{(v_{n-2}, \dots, v_1) \mid \exists v_n, v_{n-1} \in \{0, 1\} \text{ with } (v_n, v_{n-1}, \dots, v_1) \in F\}$, then $|F'| \leq |F| \leq \frac{2^n}{4f_n}$ and $F' \subseteq Q_{n-2}$. By the above, there exists an automorphism g' of Q_{n-2} with $|g'(F') \cap FC_{n-2}| \leq \frac{|F'| \cdot |FC_{n-2}|}{2^{n-2}} \leq \frac{2^n}{4f_n} \cdot \frac{f_n}{2^{n-2}} = 1$. Thus for any automorphism g of Q_n extending g' we conclude that $|g(F) \cap FC_n| \leq 1$. By Fig. 2, $\phi(2) = 1$, and therefore there exists an extension automorphism $g \in Aut(Q_n)$ of g' with $g(F) \cap FC_n = \emptyset$. Hence $\phi(n) \geq \frac{2^n}{4f_n}$. The remainder follows from the standard estimate of Fibonacci numbers. $\square$

n	1	2	3	4	5	6	7	8	9	10
$\phi(n)$	0	1	1	3	5	7	7	7	10	14
n	11	12	13	14	15	16	17	18	19	20
$\phi(n)$	14	14	20	28	28	28	40	56	56	56
n	21	22	23	24	25	26	27	28	29	30
$\phi(n)$	80	112	112	112	160	224	224	224	320	448

Fig. 3. Lower bound of $\phi(n)$ from Theorem 9 and 11 and the fact $\phi(n+1) \geq \phi(n)$

5 Upper Bound

Let us say that a subset $F \subseteq Q_n$ is *tolerable* if there exists a direct embedding g from FC_n to $Q_n \setminus F$. In other words, if we remove faulty vertices F from hypercube, there still exists a subgraph isomorphic to a Fibonacci Cube.

In this section, we search for a small set of faults that is not tolerable. We use the fact that the Fibonacci Cube of dimension n contains a subgraph isomorphic to the hypercube of dimension $\lceil n/2 \rceil$. To see this, consider a subgraph on vertices $0*0*\dots*0*$ (resp. $*0*0*\dots*0*$). The set of faults that is not tolerable for any embedding of a hypercube of dimension $\lceil n/2 \rceil$ to the hypercube of dimension n is also not tolerable for any embedding of the Fibonacci Cube of dimension n . Lemma 6 constructs such set. It consists of certain levels of hypercube.

Definition 5. For $0 \leq i \leq n$, let Q_n^i denote the i -th level of a hypercube

$$Q_n^i = \{u \in Q_n \mid w(u) = i\},$$

where $w(u)$ is the number of 1's in u (weight).

Lemma 6. Let $1 \leq m \leq n$. For every $0 \leq k \leq m$ the set of vertices

$$F = \bigcup_{l=0}^{\lfloor (n-k)/m \rfloor} Q_n^{k+lm}$$

is not tolerable for any embedding of Q_m into Q_n .

Proof. Omitted for full version.

Theorem 2. For every $n > 1$, $\phi(n) = O(2^{dn})$, for $d = (8 - 3 \log_2 3)/4 \doteq 0.82$.

Proof. A Fibonacci Cube of dimension n contains a subgraph isomorphic to the hypercube of dimension $m = \lceil n/2 \rceil$. Set $k = \lceil m/2 \rceil$. Take the set F consisting of k -th level and $(k + m)$ -th level. By Lemma 6, F is not tolerable for any direct embedding of a hypercube of dimension m and neither is tolerable for any embedding of a Fibonacci Cube of dimension n , hence $\phi(n) < |F|$.

There are four cases: $n = 4k, \dots, 4k - 3$. In all cases, we can compute

$$\begin{aligned} \phi(n) &< |F| \leq 2 \binom{4k}{k} = 2 \frac{(4k)!}{k!(3k)!} \\ &= \frac{2\sqrt{8\pi k} \left(\frac{4k}{e}\right)^{4k} \Theta(1)}{\sqrt{2\pi k} \left(\frac{k}{e}\right)^k \Theta(1) \cdot \sqrt{6\pi k} \left(\frac{3k}{e}\right)^{3k} \Theta(1)}, & \text{since } x! = \sqrt{2\pi x} \left(\frac{x}{e}\right)^x \Theta(1) \\ &= \frac{4^{4k} k^{4k} e^k e^{3k}}{3^{3k} k^k k^{3k} e^{4k}} O(1), & \text{since } \frac{4\sqrt{2\pi k} \Theta(1)}{\sqrt{2\pi k} \sqrt{6\pi k} \Theta(1) \Theta(1)} = O(1) \\ &= \left(\frac{4^4}{3^3}\right)^k O(1) = 2^{(8-3 \log_2 3)k} O(1), & \text{we have } k = \frac{n}{4} + O(1) \\ &= O(2^{dn}), & \text{where } d = \frac{8-3 \log_2 3}{4} \doteq 0.82 \quad . \quad \square \end{aligned}$$

6 NP-Complete Embedding Problems

Algorithm 1 runs in polynomial time $O(n|F|)$ with respect to size of input if faults are given by enumeration. It constructs an embedding of the Fibonacci Cube into a faulty hypercube if the number of faults is at most $p(n) = O(2^{0.25n})$.

If the number m of faults is greater than $p(n)$, we can use a simple algorithm that search all $n!2^n$ possible embeddings with time complexity $O((nm)^{c' \log n})$ for some constant c' . In case that we restrict only on 2^n possible embeddings

generated by negation of dimensions we obtain even polynomial time complexity $O(nm^5)$.

In this section, we investigate time complexity of the embedding problem under circumstance that faulty subcubes are given on input instead of faulty nodes. This problem can be applied if several tasks requiring different subcubes run at the same time. A subcube solving another task can be considered as faulty for a new task.

There is only one difference to the previous problem: faulty subcubes are given with an asterisk convention instead of enumerating of faulty nodes.

Definition 6. FibErrSubQ. *Given dimension n and a set F of $\{0, 1, *\}$ -strings of length n representing faulty subcubes of the hypercube Q_n . Is there any direct embedding $g : FC_n \rightarrow Q_n \setminus F$?*

The following theorem shows that the situation dramatically transforms.

Theorem 3. *FibErrSubQ is NP-complete.*

Proof. Consider the NP-complete 3-3-SAT ([16]) problem that is a satisfiability problem of a given formula in CNF such that

- a) any clause contains at most 3 variables,
- b) any variable is contained in at most 3 clauses.

We define a transformation of 3-3-SAT to FibErrSubQ.

Let Φ be a formula in CNF satisfying a) and b) with clauses $c_1, \dots, c_m$ and variables $x_1, \dots, x_n$. We can assume that

- 1) no clause contains both x and $\neg x$, otherwise it is always true,
- 2) every variable is used in formula Φ both in positive and negative form, otherwise it can be reduced.

From Conditions 2) and b) it follows that every variable can satisfy at most two clauses.

For each clause c_j we construct a $\{0, 1, *\}$ -string $s^j = s_1^j \dots s_m^j s_{m+1}^j \dots s_{m+n}^j$ of length $m + n$ according to a following scheme:

$$s_k^j = \begin{cases} 1 & \text{if } j = k, \\ * & \text{else} \end{cases} \quad \text{for } k = 1, \dots, m,$$

$$s_{m+i}^j = \begin{cases} 0 & \text{if } c_j \text{ contains } x_i, \\ 1 & \text{if } c_j \text{ contains } \neg x_i, \\ * & \text{if } c_j \text{ does not contain both } x_i \text{ and } \neg x_i \end{cases} \quad \text{for } i = 1, \dots, n.$$

The first m dimensions are called the clause dimensions, the rest dimensions are called variable dimensions. Let $F = \{s^j \mid j = 1, \dots, m\}$.

Now we prove that the constructed instance of FibErrSubQ has a direct embedding if and only if the given instance of 3-3-SAT has a satisfying assignment of variables.

Let 3-3-SAT have a satisfying assignment of variables $\sigma : X \rightarrow \{0, 1\}$. Denote w a witness function: $w(j) = \min\{i \mid x_i \text{ or } \neg x_i \text{ satisfies } c_j\}$.

Consider a direct embedding $g = f_\pi \circ h_A$ of the Fibonacci Cube of dimension $m + n$ into a faulty hypercube of dimension $m + n$ for $A = \{m + i \mid 1 \leq i \leq n \text{ and } \sigma(x_i) = 1\}$ and a permutation π such that $\pi(j) = \pi(m + w(j)) + 1$ or $\pi(j) = \pi(m + w(j)) - 1$ for all $j = 1, 2, \dots, m$. Since any variable can be a witness of satisfiability for at most two clauses such permutation π exists.

An automorphism $g = f_\pi \circ h_A$ maps all faulty subcubes onto subcubes containing two consecutive ones. Consequently, $g^{-1} : FC_{m+n} \rightarrow Q_{m+n} \setminus F$ is the direct embedding of the FibErrSubQ problem.

Assume that the constructed instance of the FibErrSubQ problem has a required direct embedding. Then there exist a set A and a permutation π so that the automorphism $g = f_\pi \circ h_A$ maps all faulty vertices onto vertices containing two consecutive ones. Define an assignment of variables $\sigma : X \rightarrow \{0, 1\}$ as follows

$$\sigma(x_i) = 1 \text{ if } m + i \in A \text{ else } \sigma(x_i) = 0.$$

Now, we explore dimensions that can be mapped so that we obtain two consecutive ones. Due to an asterisk convention it occurs only for pairs of fixed (non-asterisk) dimensions in the whole faulty subcube. Hence for every string s^j , there exists a pair $\{k_1, k_2\}$ of distinct non-asterisk dimensions, it means that $s_{k_1}^j \neq *$ and $s_{k_2}^j \neq *$, such that the mapping $g = f_\pi \circ h_A$ maps them to two consecutive ones. Thus $\pi(k_1) = \pi(k_2) + 1$ or $\pi(k_1) = \pi(k_2) - 1$ and $k_1 \in A$ iff $s_{k_1}^j = 0$ and $k_2 \in A$ iff $s_{k_2}^j = 0$.

At least one of k_1, k_2 must be a variable dimension (let it be k_1), the other one can be a clause dimension j or a variable dimension. Consequently, a variable x_i for $i = k_1 - m$ is a witness of satisfiability of a clause c_j because

$$\begin{aligned} \text{if } c_j \text{ contains } x_i &\Rightarrow s_{k_1=m+i}^j = 0 \Rightarrow k_1 = i + m \in A \Rightarrow \sigma(x_i) = 1, \\ \text{else } c_j \text{ contains } \neg x_i &\Rightarrow s_{k_1=m+i}^j = 1 \Rightarrow k_1 = i + m \notin A \Rightarrow \sigma(x_i) = 0. \end{aligned}$$

Therefore, every clause has a witness of satisfiability and the assignment σ is a solution of the given 3-3-SAT problem.

To see that FibErrSubQ is NP, choose nondeterministically a subset A and a permutation π and check that the automorphism $g = f_\pi \circ h_A$ maps all faulty subcubes onto subcubes containing two consecutive ones. $\square$

If we restrict FibErrSubQ only on direct embeddings generated by negation of dimensions (denote Neg-FibErrSubQ) it remains NP-complete problem.

Theorem 4. *Neg-FibErrSubQ is NP-complete.*

Proof. By a transformation of SAT, omitted for full version.

7 Conclusions

We have presented an algorithm that constructs a direct embedding of FC_n into faulty Q_n with $\Omega(2^{0.25n})$ faults. This enables us to run efficiently various parallel

algorithms with recursive design on hypercube architectures with many faulty or busy nodes ([1], [4], [5]). We have also given an upper and a lower bound on the number of tolerable faults $\Omega(2^{0.31n}) = \phi(n) = O(2^{0.82n})$.

We have proved that if we allow faulty subcubes on input, the problem of finding a direct embedding becomes NP-complete. Even we restrict on direct embeddings generated by negation of dimensions, it remains NP-complete. These results contrast with a polynomial algorithm (or nearly polynomial for a general case) that finds a direct embedding of a Fibonacci Cube into a hypercube of the same dimension if faults are given on input by enumeration.

We have investigated only faulty nodes. It might be interesting to admit also faulty links and consider embeddings of binomial trees. We think that our results can be extended to various modifications of Fibonacci Cubes such as Extended Fibonacci Cubes [15]. Remark that Algorithm 1 uses permutations only for small groups of dimensions. This indicates that some improvements are still possible.

Acknowledgments. This paper is dedicated to the memory of Ivan Havel. We wish to thank Václav Koubek for his support and encouragement. Thanks also to anonymous referee for his comments.

References

1. J. Bergum, B. Cong, and S. Sharma: Simulation of Tree Structures on Fibonacci Cubes. Proc. First Int'l Conf. Computer Comm. and Networks (1992) 279–283
2. M. Y. Chan and S. J. Lee: Fault-Tolerant Embedding of Complete Binary Trees in Hypercubes. IEEE Trans. Parallel and Distributed Systems 4 (1993) 277–288
3. M. J. Chung and W.-J. Hsu: Generalized Fibonacci Cubes. Proc. 1993 Int'l Conf. Parallel Processing 1 (1993) 299–302
4. B. Cong and S. Q. Zheng: Near-Optimal Embeddings of Trees into Fibonacci Cubes. Proc. 28th IEEE Southeastern Symp. System Theory (1996) 421–426
5. B. Cong, S. Sharma, and S. Q. Zheng: On Simulations of Linear Arrays, Rings, and 2-D Meshes on Fibonacci Cube Networks. Proc. 7th Int'l Parallel Processing Symp. (1993) 748–751
6. R. L. Graham, D. E. Knuth, and O. Patashnik: “Special numbers,” in Concrete Mathematics. Reading, Addison-Wesley, Massachusetts (1989)
7. P. Gregor: Embeddings of Special Graph Classes into Hypercubes and their Generalizations. Charles University, Prague, master thesis (1999)
8. J. Hastad, F. T. Leighton, and M. Newman: Reconfiguring a hypercube in the presence of faults. Proc. 19th Annu. ACM Symp. Theory Comput. (1987) 274–284
9. S.-J. Horng, F.-S. Jiang, T.-W. Kao: Embedding of Generalized Fibonacci Cubes in Hypercubes with Faulty Nodes. IEEE Trans. Parallel and Distributed Systems 8 (1997) 727–737
10. W.-J. Hsu: Fibonacci Cubes-A New Interconnection Topology. IEEE Trans. Parallel and Distributed Systems 4 (1993) 3–12
11. W.-J. Hsu and J. Liu: Fibonacci Codes as Formal Languages. Technical Report CPS-91-05, Michigan State University (1991)
12. F. T. Leighton: Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes Morgan Kaufmann, San Mateo, California (1992)

13. C. S. Raghavendra and P.-J. Yang: Embedding and Reconfiguration of Binary Trees in Faulty Hypercubes. *IEEE Trans. Parallel and Distributed Systems* **7** (1996) 237–245
14. N. F. Tzeng: Structural Properties of Incomplete Hypercube Computers. *Proc. 10th IEEE Int'l Conf. Distributed Computing Systems* (1990) 262–269
15. J. Wu: Extended Fibonacci Cubes. *IEEE Trans. Parallel and Distributed Systems* **8** (1997) 1203–1210
16. M. R. Garey and D. J. Johnson: *Computers and Intractability*. Bell Laboratories, New Jersey (1979)

Periodic-Like Words

Arturo Carpi¹ and Aldo de Luca²

¹ Istituto di Cibernetica del CNR, via Toiano 6, 80072 Arco Felice (NA), Italy

`arturo@arturo.cib.na.cnr.it`

² Dipartimento di Matematica dell'Università di Roma 'La Sapienza', piazzale Aldo

Moro 2, 00185 Roma, Italy

`deluca@mat.uniroma1.it`

Abstract. We introduce the notion of periodic-like word. It is a word whose longest repeated prefix is not right special. Some different characterizations of this concept are given. We derive a new condition ensuring that the greatest common divisor of two periods of a word is a period, too. Then we characterize periodic-like words having the same set of proper boxes, in terms of the important notion of root-conjugacy. Finally, some new uniqueness conditions for words, related to the maximal box theorem are given.

1 Introduction

In combinatorics on words an important role is played by periodic words [6,7]. In the finite case, a word w is periodic if $|w| \geq 2\pi_w$, where π_w is the minimal period of w . However, in the analysis of very long words such as DNA sequences the minimal period is, in general, greater and also much greater than half of the length of the word. Hence, one needs a notion more general than periodic word and such to preserve some typical features of periodic words. In this paper, we consider a large class of such words that we call *periodic-like*. This class is introduced in the frame of a new combinatorial theory of finite words which has been recently developed by the authors [2,3,5]. In this theory an essential role is played by extendable and special factors (see [1,2,3,5] and references therein). We recall that a factor u of a word w is *right extendable* if there exists a letter a such that ua is still a factor of w while u is *right special* if there exist two distinct letters a and b such that ua and ub are both factors of w . *Left extendable* and *left special* factors are symmetrically defined. A factor which is both right and left special is said to be *bispecial*.

In a previous paper [2], motivated by the search for sets of 'short' factors of a word which uniquely determine the word itself, we introduced the notion of *box*: a *proper box* of a word w is any factor of the kind asb with a and b letters and s a bispecial factor of w . Moreover, the shortest factor w which is not right (resp. left) extendable is called the *terminal* (resp. *initial*) *box*. The main result of [2], called *maximal box theorem*, states that *any word is uniquely determined by its initial and terminal boxes and by the set of its proper boxes which are maximal with respect to the factorial order*.

The lengths K_w and H_w of the terminal and initial box, respectively, as well as the least non-negative integers R_w and L_w such that there is no right special factor of length R_w and no left special factor of length L_w , give much information on the structure of the word w . For instance, the periods of a word w are not smaller than $|w| - \min\{H_w, K_w\} + 1$ [5]. Moreover, a word is uniquely determined by its factors of length $1 + \max\{R_w, K_w\}$ [2].

Words having the same sets of factors of a given word w , up to the length $1 + R_w$ were studied in [4]. In that frame, we introduced the notion of *root-conjugacy*: two words are root-conjugate if they have the same minimal period p and their prefixes of length p are conjugate.

In Sect. 3, we introduce the notion of *periodic-like* word. There exist several equivalent definitions of this notion. In particular, a word is periodic-like if its longest repeated prefix is not right special. The class of periodic-like words properly contains that of periodic words and the class of semiperiodic words introduced in [4]. However, similarly to periodic and semiperiodic words, the minimal period of a periodic-like word w attains the lower bound $|w| - H_w + 1$.

A parameter which has a great interest in the combinatorics of a periodic-like word w is the least non-negative integer R'_w such that any prefix of w of length $\geq R'_w$ is not right special. Indeed, we shall prove that a word w is periodic-like if and only if w has a period not larger than $|w| - R'_w$ or, equivalently, if and only if $R'_w < H_w$. Moreover, we shall establish the following result, which can be viewed as an extension of the theorem of Fine and Wilf (see [6]): if a word w has two periods $p, q \leq |w| - R'_w$, then $\gcd(p, q)$ is a period of w .

The notion of periodic-like word naturally appears when one studies words having the same set of maximal proper boxes. Indeed, in Sect. 4, we show that if two words have the same set of maximal proper boxes either they are both periodic-like or none of them is periodic-like. Moreover, two periodic-like words have the same set of maximal proper boxes if and only if they are root-conjugate.

In the last section, we present some uniqueness results related to the maximal box theorem.

2 Preliminaries

Let A be a finite non-empty set. We denote by A^* the free monoid generated by A . The set A is also called *alphabet* and its elements *letters*. The elements of A^* , usually called *words*, are the finite sequences of letters of A , including the *empty word*, which will be denoted by ε . We set $A^+ = A^* \setminus \{\varepsilon\}$.

A word u is a *factor* of a word w if there exist words r and s such that $w = rus$. If $w = us$, for some word s (resp. $w = ru$, for some word r), then u is called a *prefix* (resp. a *suffix*) of w . We shall denote, respectively, by $F(w)$, $\text{Pref}(w)$, and $\text{Suff}(w)$ the set of the factors, of the prefixes, and of the suffixes of w . A factor u of w is said to be *proper* if $u \neq w$.

Let w be the non-empty word

$$w = w_1 w_2 \cdots w_n,$$

with $w_i \in A$, $1 \leq i \leq n$, $n > 0$. The integer n is called the *length* of w and denoted by $|w|$. The length of the empty word is taken equal to 0.

A positive integer $p \leq n$ is a *period* of the word w if one has

$$w_i = w_{i+p}, \quad 1 \leq i \leq n - p.$$

The minimal period of w will be denoted by π_w . The prefix of w of length π_w will be called the *root* of w and denoted by r_w . A word w is said to be *periodic* if $|w| \geq 2\pi_w$. A factor u of w which is both a proper prefix and a proper suffix of w is called a *border* of w . As is well known, p is a period of w if and only if w has a border of length $|w| - p$.

Two words u and v are said to be *conjugate* if one has $u = rs$ and $v = sr$ for suitable words r and s . It is well known that conjugacy is an equivalence relation on A^* .

A factor u of w is *right extendable* (resp. *left extendable*) in w if there exists a letter a such that ua (resp. au) is still a factor of w . The factor ua (resp. au) of w is called a *right* (resp. *left*) *extension* of u in w .

A word s is called a *right* (resp. *left*) *special factor* of w if there exist two distinct letters a and b such that sa and sb (resp. as and bs) are both factors of w . A factor of w which is both right and left special is called *bispecial*.

Let w be a word. The shortest factor of w which is not right (resp. left) extendable will be called the *terminal* (resp. *initial*) *box* of w and will be denoted by k_w (resp. h_w). One can easily verify that h_w is the shortest prefix of w which has no further occurrence in w , and k_w is the shortest suffix of w which has no further occurrence in w . In particular, if $w \neq \varepsilon$, so that $h_w, k_w \neq \varepsilon$, then one can write

$$h_w = h'_w a, \quad k_w = b k'_w$$

where h'_w and k'_w are, respectively, the longest repeated prefix and the longest repeated suffix of w and a, b are letters. We remark that a factor u of w is right (resp. left) extendable if and only if $k_w \notin \text{Suff}(u)$ (resp. $h_w \notin \text{Pref}(u)$).

A *proper box* of w is any factor of w of the kind

$$asb$$

with a and b letters and s a bispecial factor of w . A proper box is called *maximal* if it is not a factor of another proper box. We shall denote by $\mathcal{B}_w$ the set of the maximal proper boxes of a word w .

For any word w , we shall consider the parameters $H_w = |h_w|$ and $K_w = |k_w|$. Moreover, we shall denote by R_w the minimal non-negative integer such that there is no right special factor of w of length R_w and by L_w the minimal non-negative integer such that there is no left special factor of w of length L_w .

Parameters H_w, K_w, R_w , and L_w are related to the minimal period of w by the following relation (see [4,5])

$$\pi_w \geq |w| - \min\{H_w, K_w\} + 1 \geq \max\{R_w, L_w\} + 1. \quad (1)$$

3 Periodic-Like Words

A non-empty word w is called *periodic-like* if h'_w is not a right special factor of w . The reason for this name will appear clear in the sequel (see Proposition 4).

The following lemma, whose proof we omit for the sake of brevity, holds.

Lemma 1. *Let w be a periodic-like word. Then*

1. h'_w has no internal occurrences in w ,
2. $h'_w = k'_w$,
3. k'_w is not left special in w ,
4. $\pi_w = |w| - H_w + 1 = |w| - K_w + 1$,
5. $w = r_w h'_w$.

Lemma 2. *Let w be a non-empty word. The following conditions are equivalent.*

1. w is periodic-like,
2. h'_w has no internal occurrences in w ,
3. k'_w is not left special in w ,
4. k'_w has no internal occurrences in w ,
5. the maximal border of w has no internal occurrences.

Proof. We prove the equivalence of 1., 2., and 5. For symmetry reasons, 3., 4., and 5. will be also equivalent.

1. $\Rightarrow$ 5. If w is periodic-like, then, by Lemma 1, h'_w has no internal occurrences in w and $h'_w = k'_w$. Thus h'_w is the maximal border of w and it has no internal occurrences in w .

5. $\Rightarrow$ 2. Since any border of w is a prefix of h'_w , then h'_w cannot have an internal occurrence in w .

2. $\Rightarrow$ 1. Since h'_w has no internal occurrences in w , then its only right extension in w will be h_w . Thus, h'_w cannot be a right special factor of w . $\square$

Let w be a word. We denote by R'_w the least non-negative integer such that no prefix of w of length $\geq R'_w$ is a right special factor of w . In a similar way, we denote by L'_w the least non-negative integer such that no suffix of w of length $\geq L'_w$ is a left special factor of w . Notice that

$$R'_w \leq \min\{H_w, R_w\}, \quad L'_w \leq \min\{K_w, L_w\}. \quad (2)$$

Example 1. Let $w = abbbab$. One has $H_w = K_w = L_w = R_w = 3$, $R'_w = 1$, $L'_w = 2$. Since $h'_w = ab$ is not right special, then w is periodic-like and $\pi_w = |w| - H_w + 1 = 4$. The word $v = aabaacaa$ is not periodic-like since $h'_v = aa$ is right special. However, $H_v = K_v = 3$ and $\pi_v = 6 = |v| - H_v + 1$.

Proposition 1. *Let w be a non-empty word. The following conditions are equivalent.*

1. w is periodic-like,
2. $R'_w < H_w$,
3. w has a period $p \leq |w| - R'_w$,
4. $L'_w < K_w$,
5. w has a period $p \leq |w| - L'_w$.

Proof. 1. $\implies$ 2. If w is periodic-like, then h'_w is not right special. Since any prefix of length $\geq H_w$ is not right special, it follows $R'_w < H_w$.

2. $\implies$ 1. If $R'_w < H_w$, then $|h'_w| \geq R'_w$, so that h'_w is not right special.

1. $\implies$ 3. By Lemma 1, w has the minimal period $\pi_w = |w| - H_w + 1$. By the implication 1. $\implies$ 2., it follows $\pi_w \leq |w| - R'_w$.

3. $\implies$ 1. Let p be a period of w such that $p \leq |w| - R'_w$. Since by (1), $p \geq |w| - H_w + 1$, it follows $R'_w < H_w$.

Hence, Conditions 1., 2., and 3. are equivalent. Since, by Lemma 1, w is periodic-like if and only if k'_w is not left special, the equivalence of Conditions 1., 4., and 5. can be proved in a symmetrical way. $\square$

Let us recall (see [4]) that a word w is called *semiperiodic* if $R_w < H_w$. If w is semiperiodic, then $R'_w \leq R_w < H_w$. Thus, by the previous proposition, any semiperiodic word is periodic-like. However, the converse is not generally true as shown by the following example.

Example 2. Let w be the word $w = ab^n a$, with $n \geq 2$. In such a case, $h'_w = a$ is not a right special factor of w , so that w is periodic-like. However, $R_w = n \geq 2 = H_w$, so that w is not semiperiodic.

Let us observe that a periodic word w is semiperiodic and hence periodic-like. Indeed, in a periodic word w the prefix of length π_w is a repeated factor so that $\pi_w < H_w$. From (1) it follows $\pi_w > R_w$. This implies that w is semiperiodic. Moreover, by Condition 4. of Lemma 1, $H_w = K_w$. Hence, for a periodic word w , one has:

$$H_w = K_w > \pi_w > R_w. \quad (3)$$

By (1), one has also $\pi_w > L_w$. Actually, as proved in [4], in a semiperiodic word $L_w = R_w$. We remark that a semiperiodic word is not in general a periodic word. For instance, let $w = abaab$. One has $H_w = 3$, $R_w = 2$, $|w| = 5$, and $\pi_w = 3$.

The following proposition gives a condition assuring that a period of a prefix of a word can be 'extended' to the entire word.

Proposition 2. *If a word w has a prefix of period p and length $p + R'_w$, then p is a period of w .*

Proof. Set $w = a_1 a_2 \cdots a_n$, with $a_i \in A$, $1 \leq i \leq n$. Let us assume that p is not a period of w . Then there is a minimal i such that

$$1 \leq i \leq n - p \quad \text{and} \quad a_i \neq a_{i+p}. \quad (4)$$

One derives

$$\begin{aligned} a_1 \cdots a_{i-1} &= a_{1+p} \cdots a_{i-1+p} = u, \\ ua_i, ua_{i+p} &\in F(w). \end{aligned}$$

Thus, u is a right special prefix of w , and, therefore,

$$i = |u| + 1 \leq R'_w.$$

In view of (4), we conclude that p is not a period of $a_1 \cdots a_{p+R'_w}$. $\square$

Proposition 3. *If a word w has two periods $p, q \leq |w| - R'_w$, then w has also the period $d = \gcd(p, q)$.*

Proof. We can assume, with no loss of generality, that $p < q$. Set $w = a_1 a_2 \cdots a_n$, with $a_i \in A$, $1 \leq i \leq n$. For $1 \leq i \leq R'_w$, one has $i + q \leq |w| = n$ and

$$a_i = a_{i+q} = a_{i+q-p}.$$

Thus, $a_1 a_2 \cdots a_{R'_w+q-p}$ has the period $q-p$. By the preceding proposition, $q-p$ is a period of w . By an inductive argument on $\max\{p, q\}$ (or by using the theorem of Fine and Wilf, see [6]), w has the period $\gcd(p, q-p) = d$. $\square$

From the preceding proposition one can easily derive the theorem of Fine and Wilf for finite words [6].

Theorem 1. *Let w be a word having two periods p and q and length*

$$|w| \geq p + q - \gcd(p, q). \quad (5)$$

Then w has the period $\gcd(p, q)$.

Proof. It is well known that one can always reduce himself to consider only the case when $\gcd(p, q) = 1$. Since, by (1) and (2), $p, q \geq R_w + 1 \geq R'_w + 1$, one has

$$|w| \geq p + q - 1 \geq q + R'_w \quad \text{and} \quad |w| \geq p + q - 1 \geq p + R'_w.$$

This implies $p, q \leq |w| - R'_w$, so that the conclusion follows from Proposition 3. $\square$

We remark [4] that if a word w has two distinct periods p and q and $|w| \geq p + q - \gcd(p, q)$, then w has to be semiperiodic and then periodic-like. However, there are words w having two periods p and q such that $p, q \leq |w| - R'_w$ but $|w| < p + q - \gcd(p, q)$ (for instance, the word $w = abababa$ has periods $p = 4$ and $q = 6$, $R'_w = 1$, and $p + q - d = 8 > |w|$). Thus Proposition 3 is a more general formulation of the theorem of Fine and Wilf which takes into account some ‘structural’ properties of the word.

4 Root-Conjugacy

We say that two words $f, g \in A^*$ are *root-conjugate* if their roots are conjugate. It is evident that root-conjugacy is an equivalence relation in A^* .

Example 3. Let $f = abbab$, $g = bbab$. Then $r_f = abb$ and $r_g = bba$ are conjugate, so that f and g are root-conjugate.

We remark explicitly that conjugate words are not necessarily root-conjugate. For instance, the conjugate words $abba$ and $aabb$ are not root-conjugate, since they have distinct minimal periods.

The following proposition shows that any periodic-like word can be prolonged in a periodic word having the same root and the same set of proper boxes.

Proposition 4. *Let f be a periodic-like word. Then there exists a periodic word f_0 such that*

$$f \in \text{Pref}(f_0)$$

and f and f_0 have the same root and the same set of proper boxes.

Proof. If f is periodic, then there is nothing to prove. Let us then suppose that f is not periodic. Since f is periodic-like, we can write:

$$f = h'_f u h'_f, \quad \text{with } u \in A^+.$$

Let us set

$$f_0 = h'_f u h'_f u = f u.$$

By Lemma 1, one has $r_f = h'_f u$ and then $\pi_f = |h'_f u|$. Moreover, f_0 has the period π_f so that $\pi_{f_0} \leq \pi_f$ and $\pi_f \leq \pi_{f_0}$ since f is a factor of f_0 . Thus, $\pi_f = \pi_{f_0}$ that implies $r_f = r_{f_0} = h'_f u$.

Since f is a prefix of f_0 , any proper box of f is, trivially, a proper box of f_0 . Then, let us prove the converse.

Let $\alpha = asb$ be a proper box of f_0 , with $a, b \in A$ and s a bispecial factor of f_0 . Let us prove that $h'_f \notin F(s)$. Indeed, by Lemma 1, h'_f has no internal occurrence in $f = h'_f u h'_f$ and, therefore, has exactly two occurrences in f_0 , one of which is initial, whereas s has at least two non-initial occurrences in f_0 , since s is a left special factor of f_0 .

Since h'_f is not a factor of s , then $\alpha = asb$ occurs either in the prefix $h'_f u h'_f$ of f_0 or in the suffix $h'_f u$ of f_0 . Thus $\alpha \in F(f)$. Let us now prove that any right or left extension of s in f_0 is a factor of f . Indeed, since h'_f is not a factor of s , this extension has to be a factor either of the prefix $h'_f u h'_f$ of f_0 or of the suffix $h'_f u$ of f_0 . In any case, it is a factor of f . Thus s is a bispecial factor of f and asb is a proper box of f . This proves that any proper box of f_0 is a proper box of f . $\square$

The following important lemma holds [4].

Lemma 3. *Let f and g be words such that $\text{alph}(f) \cap \text{alph}(g) \neq \emptyset$. If*

$$\mathcal{B}_f \subseteq F(g), \quad \mathcal{B}_g \subseteq F(f),$$

then

$$F(g) \subseteq F(f) \cup A^+ h_f A^* \cup A^* k_f A^+.$$

Proposition 5. *Let f be a periodic-like word. If g is a word such that $\text{alph}(f) \cap \text{alph}(g) \neq \emptyset$ and*

$$\mathcal{B}_f \subseteq F(g), \quad \mathcal{B}_g \subseteq F(f),$$

then g is periodic-like and root-conjugate of f .

Proof. Let us first verify that g is periodic-like. By Proposition 4, we can assume, with no loss of generality, that f is periodic. Consequently, by (3), $H_f = K_f > R_f$. Assume that g is not periodic-like. Then h'_g is a right special factor of g . This implies that there exist letters a and b , $a \neq b$, such that

$$h_g = h'_g a \quad \text{and} \quad h'_g b \in F(g).$$

If $H_g \geq H_f$, then there exists a suffix t of h'_g such that

$$ta, tb \in F(g), \quad |ta| = |tb| = H_f = K_f.$$

By Lemma 3 one derives that $ta, tb \in F(f)$. Thus t is right special in f , so that $R_f \geq |t| + 1 = H_f$, which is a contradiction.

If, on the contrary, $H_g < H_f$, then one has $|h'_g a| = |h'_g b| < H_f = K_f$ and therefore, by Lemma 3,

$$h'_g a, h'_g b \in F(f).$$

Denote by s the longest word such that $sh'_g a, sh'_g b \in F(f)$. Since sh'_g is right special in f , one has

$$|sh'_g a| = |sh'_g b| \leq R_f < H_f.$$

Thus, $sh'_g a$ and $sh'_g b$ are left extendable in f , i.e., there exist letters c and d such that

$$csh'_g a, dsh'_g b \in F(f).$$

Moreover, $c \neq d$ by the maximality of $|s|$. We conclude that $csh'_g a$ is a proper box of f and, hence, a factor of g . This yields a contradiction, for $h'_g a = h_g$ is not left extendable in g . This proves that g is periodic-like.

Now, let us prove that f and g are root-conjugate. By Proposition 4 we can assume, with no loss of generality, that also g is periodic. Since by (3), $H_f = K_f \geq 1 + \pi_f$, by Lemma 3 all the factors of g of length $1 + \pi_f$ are factors of f and, therefore, they have the period π_f . From this, one easily derives that g has the period π_f . By a symmetrical argument, π_g is a period of f and, therefore, $\pi_f = \pi_g$. Moreover, r_g is a factor of f of length π_f and, therefore, it is conjugate of r_f . We conclude that f and g are root-conjugate. $\square$

Proposition 6. *Let f and g be root-conjugate and periodic-like words. Then*

$$\mathcal{B}_f = \mathcal{B}_g.$$

Proof. By Proposition 4, we can reduce ourselves to the case that f and g are periodic. Consequently, f and g have the same set of factors of length $\pi_f = \pi_g$.

Indeed, since f and g are root-conjugate, this set coincides with the conjugacy class of r_f and r_g .

Since by (1), $R_f, L_f < \pi_f = \pi_g$, any right (resp. left) special factor of f is also a right (resp. left) special factor of g . From this one easily derives that any maximal proper box of f is also a maximal proper box of g , i.e., $\mathcal{B}_f \subseteq \mathcal{B}_g$. In a symmetrical way, one derives that $\mathcal{B}_g \subseteq \mathcal{B}_f$, that concludes the proof. $\square$

The following example shows that the assumption of being periodic-like for the words in the previous propositions cannot be eliminated.

Example 4. The words $f = a^2b^3$ and $g = a^3b^2$ are not periodic-like. One has $\mathcal{B}_f = \mathcal{B}_g = \{aa, ab, bb\}$ but f and g are not root-conjugate.

The words $f = aba$, $g = ab$ are root-conjugate but $\mathcal{B}_f = \{ab, ba\} \neq \{ab\} = \mathcal{B}_g$. Notice that f is periodic-like but g is not.

We can summarize the previous propositions in the following theorem.

Theorem 2. *Let f and g be words. If f is periodic-like, then the following conditions are equivalent:*

1. $\text{alph}(f) \cap \text{alph}(g) \neq \emptyset$, $\mathcal{B}_f \subseteq F(g)$, and $\mathcal{B}_g \subseteq F(f)$,
2. $\text{alph}(f) \cap \text{alph}(g) \neq \emptyset$ and $\mathcal{B}_f = \mathcal{B}_g$,
3. g is periodic-like and f and g are root-conjugate.

Proof. $1 \Rightarrow 3$. By Proposition 5.

$3 \Rightarrow 2$. Since f and g are root-conjugate, then $\text{alph}(f) \cap \text{alph}(g) \neq \emptyset$. By Proposition 6, one has $\mathcal{B}_f = \mathcal{B}_g$.

$2 \Rightarrow 1$. Trivial. $\square$

Remark 1. When f is periodic-like, one has $(\mathcal{B}_f \subseteq F(g), \mathcal{B}_g \subseteq F(f)) \iff \mathcal{B}_f = \mathcal{B}_g$, even if $\text{alph}(f) \cap \text{alph}(g) = \emptyset$. Indeed, in this latter case, both conditions are satisfied if and only if $\mathcal{B}_f = \mathcal{B}_g = \emptyset$. Let us observe that in the preceding theorem one cannot drop the hypothesis that f is periodic-like. This is shown by the following example. Let $f = abcabdab$ and $g = bcabda$. Then f is not periodic-like and $\mathcal{B}_f = \{bc, cabd, da\} \subseteq F(g)$, $\mathcal{B}_g = \{bc, ca, ab, bd, da\} \subseteq F(f)$, but $\mathcal{B}_f \neq \mathcal{B}_g$. Moreover, g is root-conjugate of f but it is not periodic-like.

5 Boxes and Uniqueness Conditions

In the previous section, we have shown that the root-conjugacy class of a periodic-like word, with the only exception of a trivial case, is uniquely determined by the set of its maximal proper boxes. In this section we show that some further information, such as, e.g., the knowledge of a suitable prefix (or suffix) and of the length of the word determines uniquely the word itself.

Theorem 3. *Let f and g be two words such that*

1. $|f| = |g|$,
2. f and g have a common prefix of length $\max\{1, R'_f, R'_g\}$,
3. $\mathcal{B}_f \subseteq F(g)$, $\mathcal{B}_g \subseteq F(f)$.

Then $f = g$.

Proof. With no loss of generality, we can suppose that $H_f \geq H_g$. Assume, by contradiction, that $f \neq g$. Since $|f| = |g|$, then there exist words $v, r, s \in A^*$ and letters $a, b \in A$, with $a \neq b$, such that

$$f = var, \quad g = vbs.$$

By Condition 2., $|v| \geq 1$, so that $\text{alph}(f) \cap \text{alph}(g) \neq \emptyset$. By Lemma 3, one has

$$vb \in F(f) \cup A^+ h_f A^* \cup A^* k_f A^+. \quad (6)$$

If $vb \in F(f)$, then v is a right special prefix of f and, therefore, $|va| = |vb| \leq R'_f \leq \max\{1, R'_f, R'_g\}$, which contradicts Condition 2.

If $vb \in A^* k_f A^+$, then one has also $va \in A^* k_f A^+$ and, therefore, $f \in A^* k_f A^+$, which is a contradiction, since k_f cannot be right extendable in f .

Thus, in view of (6), one has necessarily $vb \in A^+ h_f A^*$. Moreover, $|v| \geq H_f \geq H_g$ and, consequently, h_f and h_g are both prefixes of v . Since $H_f \geq H_g$, the word h_g is a prefix of h_f , so that $vb \in A^+ h_g A^*$. This yields a contradiction, since h_g cannot be left extendable in the factor vb of g . $\square$

By an argument very similar to that of the preceding proof, one can prove the following extension of the maximal box theorem.

Theorem 4. *Let f and g be two words such that*

1. f and g have a common prefix of length $\max\{R'_f, R'_g\}$,
2. f and g have a common suffix of length $\max\{K'_f, K'_g\}$,
3. $\mathcal{B}_f \subseteq F(g)$, $\mathcal{B}_g \subseteq F(f)$.

Then $f = g$.

Let us remark that Condition 2. in the previous theorem cannot be replaced by the condition that f and g have a common suffix of length $\max\{L'_f, L'_g\}$. Indeed, the two words $f = abcabcabcab$ and $g = abcabcab$ satisfy Conditions 1. and 3. and they have the same suffix of length $\max\{L'_f, L'_g\} = 1$.

References

1. Carpi, A., de Luca, A.: Words and Repeated Factors. Séminaire Lotharingien de Combinatoire **B421** (1998) 24 pp.
2. Carpi, A., de Luca, A.: Words and Special Factors. Preprint 98/33, Dipartimento di Matematica dell'Università di Roma "La Sapienza" (1998), Theor. Comput. Sci. (to appear)

3. Carpi, A., de Luca, A.: Repetitions and Boxes in Words and Pictures. In: Karhumäki, J., Maurer, H., Păun, G., Rozenberg, G. (eds.): *Jewels Are Forever*, Springer-Verlag, Berlin (1999) 295–306
4. Carpi, A., de Luca, A.: Semiperiodic words and root-conjugacy. Preprint 10/2000, Dipartimento di Matematica dell'Università di Roma "La Sapienza" (2000)
5. de Luca, A.: On the Combinatorics of Finite Words. *Theor. Comput. Sci.* **218** (1999) 13–39
6. Lothaire, M.: *Combinatorics on Words*. Addison-Wesley, Reading, MA (1983)
7. Lothaire, M.: *Algebraic Combinatorics on Words*. Cambridge University Press, Cambridge (to appear)

The Monadic Theory of Morphic Infinite Words and Generalizations

Olivier Carton¹ and Wolfgang Thomas²

¹ Institut Gaspard Monge and CNRS

Université de Marne-la-Vallée

F-77454 Marne-la-Vallée Cedex 2, France

Olivier.Carton@univ-mlv.fr

<http://www-igm.univ-mlv.fr/~carton/>

² Lehrstuhl für Informatik VII, RWTH Aachen

D-52056 Aachen, Germany

thomas@informatik.rwth-aachen.de

<http://www-i7.informatik.rwth-aachen.de/~thomas/>

Abstract. We present new examples of infinite words which have a decidable monadic theory. Formally, we consider structures $\langle \mathbb{N}, <, P \rangle$ which expand the ordering $\langle \mathbb{N}, < \rangle$ of the natural numbers by a unary predicate P ; the corresponding infinite word is the characteristic 0-1-sequence x_P of P . We show that for a morphic predicate P the associated monadic second-order theory $\text{MTh}\langle \mathbb{N}, <, P \rangle$ is decidable, thus extending results of Elgot and Rabin (1966) and Maes (1999). The solution is obtained in the framework of semigroup theory, which is then connected to the known automata theoretic approach of Elgot and Rabin. Finally, a large class of predicates P is exhibited such that the monadic theory $\text{MTh}\langle \mathbb{N}, <, P \rangle$ is decidable, which unifies and extends the previously known examples.

1 Introduction

In this paper we study the following decision problem about a fixed ω -word x :

(Acc_x) Given a Büchi automaton $\mathcal{A}$, does $\mathcal{A}$ accept x ?

If the problem (Acc_x) is decidable, this means intuitively that one can use x as external oracle information in nonterminating finite-state systems and still keep decidability results on their behaviour. We solve this problem for a large class of ω -words, the so-called *morphic words* and some generalizations, complementing and extending results of Elgot and Rabin [8] and Maes [10].

The problem (Acc_x) is motivated by a logical decision problem regarding monadic theories, starting from Büchi's theorem [6] on the equivalence between the monadic second-order theory $\text{MTh}\langle \mathbb{N}, < \rangle$ of the linear order $\langle \mathbb{N}, < \rangle$ and ω -automata (more precisely: Büchi automata). Büchi used this reduction of formulas to automata to show that $\text{MTh}\langle \mathbb{N}, < \rangle$ is decidable. The decidability proof is based on the fact that a sentence ϕ of the monadic second-order language of

$\langle \mathbb{N}, < \rangle$ can be converted into an input-free Büchi automaton $\mathcal{A}$ such that ϕ holds in $\langle \mathbb{N}, < \rangle$ iff $\mathcal{A}$ admits some successful run; the latter is easily checked.

It was soon observed that Büchi's Theorem is applicable also in a more general situation, regarding expansions of $\langle \mathbb{N}, <, P \rangle$ of the structure $\langle \mathbb{N}, < \rangle$ by a fixed predicate $P \subseteq \mathbb{N}$. Here one starts with a formula $\phi(X)$ with one free set variable and considers an equivalent Büchi automaton $\mathcal{A}$ over the input alphabet $\mathbb{B} = \{0, 1\}$: The formula $\phi(X)$ is true in $\langle \mathbb{N}, < \rangle$ with P as interpretation of X iff $\mathcal{A}$ accepts the characteristic word x_P over $\mathbb{B}$ associated with P (the i -th letter of x_P is 1 iff $i \in P$). In other words: The theory $\text{MTh}\langle \mathbb{N}, <, P \rangle$ is decidable if one can determine, for any given Büchi automaton $\mathcal{A}$, whether $\mathcal{A}$ accepts x_P . Elgot and Rabin [8] followed this approach via the decision problem (Acc_{x_P}) and found several interesting predicates P such that $\text{MTh}\langle \mathbb{N}, <, P \rangle$ is decidable, among them the set of factorial numbers $n!$, the set of k -th powers n^k for any fixed k , and the set of k -powers k^n for any fixed k .

Elgot and Rabin (and later also Siefkes [13]) used an automata theoretic analysis, which we call *contraction method*, for the solution of the decision problem (Acc_{x_P}) . The idea is, given a Büchi automaton $\mathcal{A}$, to contract the 0-sections in x_P in such a way that an *ultimately periodic* ω -word β is obtained which is accepted by $\mathcal{A}$ iff x_P is accepted by $\mathcal{A}$. For the ultimately periodic word β , one can easily decide whether it is accepted by $\mathcal{A}$.

More recently, A. Maes [10] studied “morphic predicates”, which are obtained by iterative application of a morphism on words (see Sect. 2 for definitions). The morphic predicates include examples (for instance, the predicate of the Fibonacci numbers) which do not seem to be accessible by the Elgot-Rabin method. Maes proved that for any morphic predicate P , the *first-order* theory $\text{FTh}\langle \mathbb{N}, <, P \rangle$ is decidable, and he also introduced appropriate (although special) versions of morphic predicates of higher arity. It remained open whether for each unary morphic predicate P , the monadic theory $\text{MTh}\langle \mathbb{N}, <, P \rangle$ is decidable.

In the present paper we answer this question positively, based on a new (and quite simple) semigroup approach to the decision problem (Acc_x) : In Sect. 2 we show that for morphic predicates P the problem (Acc_{x_P}) and hence the monadic theory $\text{MTh}\langle \mathbb{N}, <, P \rangle$ is decidable. Interesting example predicates covered by this result are the Fibonacci predicate (consisting of all Fibonacci numbers) and the Thue-Morse word predicate (consisting of those numbers whose binary expansion has an even number of 1's).

In the second part of the paper, we embed the semigroup approach into the framework of the contraction method. This method is shown to be applicable to predicates which we call “residually ultimately periodic”. We prove two results: Each morphic predicate is residually ultimately periodic, and a certain class of residually ultimately periodic predicates shares strong closure properties, among them closure under sum, product, and exponentiation. This allows to obtain many example predicates P for which the monadic theory $\text{MTh}\langle \mathbb{N}, <, P \rangle$ is decidable.

It should be noted that for certain concrete applications (such as for the Fibonacci predicate) the semigroup approach is much more convenient than an

explicit application of the contraction method, and only an analysis in retrospect reveals the latter possibility. Also morphic predicates like the Thue-Morse word predicate are not approachable directly by the contraction method, because there are no long sections of 0's or 1's.

Let us finally comment on example predicates P where the corresponding Büchi acceptance problem (Acc_{x_P}) and hence $\text{MTh}\langle\mathbb{N}, <, P\rangle$ is undecidable, and give some comments on unsettled cases and on related work.

First we recall a simple recursive predicate P such that $\text{MTh}\langle\mathbb{N}, <, P\rangle$ is undecidable. For this, consider a non-recursive, recursively enumerable set Q of positive numbers, say with recursive enumeration $m_0, m_1, m_2, \dots$. Define $P = \{n_0, n_1, n_2, \dots\}$ by $n_0 = 0$ and $n_{i+1} = n_i + m_i$. Then P is recursive but even the first-order theory $\text{FTh}\langle\mathbb{N}, <, P\rangle$ is undecidable: We have $k \in Q$ iff for some element x of P , the element $x + k$ is the next element after x in P , a condition which is expressible by a sentence ϕ_k of the first-order language of $\langle\mathbb{N}, <, P\rangle$. Büchi and Landweber [7] and Thomas [14] determined the recursion theoretic complexity of the theories $\text{MTh}\langle\mathbb{N}, <, P\rangle$ for recursive P ; it turns out, for example, that for recursive P the theory $\text{MTh}\langle\mathbb{N}, <, P\rangle$ is truth-table-reducible to a complete Σ_2 -set, and that this bound cannot be improved. In [15] it was shown that for each predicate P , the full monadic theory $\text{MTh}\langle\mathbb{N}, <, P\rangle$ is decidable iff the weak monadic theory $\text{WMTh}\langle\mathbb{N}, <, P\rangle$ is (where all set quantifiers are assumed to range only over finite sets). However, there are examples P such that the first-order theory $\text{FTh}\langle\mathbb{N}, <, P\rangle$ is decidable but $\text{WMTh}\langle\mathbb{N}, <, P\rangle$ is undecidable ([14]).

In the present paper we restrict ourselves to expansions of $\langle\mathbb{N}, <\rangle$ by unary predicates. For expansions $\langle\mathbb{N}, <, f\rangle$ by unary functions f , the undecidability of $\text{MTh}\langle\mathbb{N}, <, f\rangle$ arises already in very simple cases, like for the doubling function $n \mapsto 2n$. For a survey on these theories see [11].

On the other hand, it seems hard to provide mathematically natural recursive predicates P for which $\text{MTh}\langle\mathbb{N}, <, P\rangle$ is undecidable. A case where the question is unsettled is the prime number predicate. As remarked already by Büchi and Landweber in [7], a decidability proof should be difficult because it would in principle answer unsolved number theoretic problems like the twin prime hypothesis (which states that infinitely many pairs $(p, p + 2)$ of primes exist). The only known result is that $\text{MTh}\langle\mathbb{N}, <, P\rangle$ is decidable under the linear case of Schinzel's hypothesis [3].

The authors would like to thank Jorge Almeida, Jean-Paul Allouche, Jean Berstel and Jacques Désarménien for very interesting suggestions and comments.

2 Büchi Automata over Morphic Predicates

A *morphism* τ from A^* to itself is an application such that the image of any word $u = a_1 \dots a_n$ is the concatenation $\tau(a_1) \dots \tau(a_n)$ of the images of its letters. A morphism is then completely defined by the images of the letters. In the sequel, we describe morphisms by just specifying the respective images of the letters as in the following example: $\tau : a \mapsto ab, b \mapsto ccb, c \mapsto c$.

Let A be a finite alphabet and let τ be a morphism from A^* to itself. For any integer n , we denote by τ^n the composition of n copies of τ . Let $(x_n)_{n \geq 0}$ be the sequence of finite words defined by $x_n = \tau^n(a)$ for any integer n . If the first letter of $\tau(a)$ is a , then each word x_n is a prefix of x_{n+1} . If furthermore the sequence of length $|x_n|$ is not bounded, the sequence $(x_n)_{n \geq 0}$ converges to an infinite word x which is denoted by $\tau^\omega(a)$. The word x is a *fixed point* of the morphism since it satisfies $x = \tau(x)$.

Example 1. Let consider the morphism τ given by $\tau : a \mapsto ab, b \mapsto ccb, c \mapsto c$. The words $x_n = \tau^n(a)$ for $n = 1, 2, 3, 4$ are respectively equal to

$$\begin{aligned} \tau(a) &= ab & \tau^3(a) &= abccbccccb \\ \tau^2(a) &= abccb & \tau^4(a) &= abccbccccbcccccb \end{aligned}$$

It can be easily proved by induction on n that $\tau^{n+1}(a) = abc^2bc^4b \dots c^{2n}b$. Therefore, the fixed point $\tau^\omega(a)$ is equal to the infinite word $abc^2bc^4bc^6bc^8 \dots$.

An infinite word x over B is said to be *morphic* if there is a morphism τ from A^* to itself and a morphism σ from A^* to B^* such that $x = \sigma(\tau^\omega(a))$. In the sequel, the alphabet B is often the alphabet $\mathbb{B} = \{0, 1\}$ and the morphism σ is letter to letter. The *characteristic word* of a predicate P over the set $\mathbb{N}$ of non-negative integers is the infinite word $x_P = (b_n)_{n \geq 0}$ over the alphabet $\mathbb{B}$ defined by $b_n = 1$ iff $n \in P$ and $b_n = 0$ otherwise. A predicate is said to be *morphic* iff its characteristic word is morphic.

Example 2. Consider the morphism τ introduced in the preceding example and the morphism σ given by $\sigma : a \mapsto 1, b \mapsto 1, c \mapsto 0$. The morphic word $\sigma(\tau^\omega(a)) = 1100100001 \dots$ is actually the characteristic word of the predicate $P = \{n^2 \mid n \in \mathbb{N}\}$. This can be easily proved using the equality $(n+1)^2 = \sum_{k=0}^n 2k+1$. The square predicate is therefore morphic.

Example 3. Consider the morphism τ from $\mathbb{B}^*$ to $\mathbb{B}^*$ given by $\tau : 0 \mapsto 01, 1 \mapsto 10$. The fixed point $\tau^\omega(1) = 100101100110 \dots$ is the characteristic word of the predicate P with $n \in P$ iff the binary expansion of n contains an even number of 1. The fixed point $\tau^\omega(1)$ is the well-known Thue-Morse word (see e.g. [4]).

Recall that a *Büchi automaton* is an automaton $\mathcal{A} = (Q, E, I, F)$ where Q is a finite set of states, $E \subseteq Q \times A \times Q$ is the set of transitions and I and F are the sets of initial and final states. A path is successful if it starts in an initial state and goes infinitely often through a final state. An infinite word is accepted if it is the label of a successful path. We refer the reader to [16] for a complete introduction. Now we can state the main result and its formulation in the context of monadic theories:

Theorem 1. *Let $x = \sigma(\tau^\omega(a))$ be a fixed morphic word where $\tau : A^* \rightarrow A^*$ and $\sigma : A^* \rightarrow B^*$ are morphisms. For any Büchi automaton $\mathcal{A}$, it can be decided whether x is accepted by $\mathcal{A}$.*

Corollary 1. *For any unary morphic predicate P , the monadic second-order theory of $\langle \mathbb{N}, <, P \rangle$ is decidable.*

Proof. (of Theorem 1) Let $\mathcal{A} = (Q, E, I, F)$ be a Büchi automaton. Define the following equivalence relation $\equiv$ over A^* (already introduced by Büchi in [6]):

$$u \equiv u' \stackrel{\text{def}}{\iff} \forall p, q \in Q \quad \begin{cases} p \xrightarrow{u} q \iff p \xrightarrow{u'} q \\ p \xrightarrow[F]{u} q \iff p \xrightarrow[F]{u'} q \end{cases}$$

where $p \xrightarrow{u} q$ means that there is a path from p to q labeled by u and $p \xrightarrow[F]{u} q$ means that there is a path from p to q labeled by u which hits some final state.

Denote by π the projection from A^* to $A^*/\equiv$ which maps each word to its equivalence class.

The equivalence relation $\equiv$ is a congruence of finite index. So a product on the congruence classes can be defined which turns the set $A^*/\equiv$ into a finite semigroup. The projection π is then a morphism from A^* onto $A^*/\equiv$.

The following fact on the congruence $\equiv$ is well-known: Suppose that the two infinite words x and x' can be factorized $x = u_0 u_1 u_2 \dots$ and $x' = u'_0 u'_1 u'_2 \dots$ such that $u_k \equiv u'_k$ for any $k \geq 0$; then x is accepted by $\mathcal{A}$ iff x' is accepted by $\mathcal{A}$.

Since a is the first letter of $\tau(a)$, the word $\tau(a)$ is equal to au for some nonempty finite word u . It may be easily verified by induction on n that for any integer n , one has $\tau^{n+1}(a) = au\tau(u)\tau^2(u) \dots \tau^n(u)$. The word $x = \sigma(\tau^\omega(a))$ can be factorized $x = u_0 u_1 u_2 \dots$ where $u_0 = \sigma(au)$ and $u_n = \sigma(\tau^n(u))$ for $n \geq 1$.

We claim that there are two positive integers n and p such that for any $k \geq n$, the relation $u_k \equiv u_{k+p}$ holds, in other words $\pi(u_k) = \pi(u_{k+p})$. A morphism from A^* into a semigroup is completely determined by the images of the letters; so there are only finitely many morphism from A^* into the finite semigroup $A^*/\equiv$. This implies that there are two positive integers n and p such that $\pi \circ \sigma \circ \tau^n = \pi \circ \sigma \circ \tau^{n+p}$. This implies that $\pi \circ \sigma \circ \tau^k = \pi \circ \sigma \circ \tau^{k+p}$ for any k greater than n , and thus $u_k \equiv u_{k+p}$. Note that these two integers n and p can be effectively computed: It suffices to check that $\sigma(\tau^n(b)) \equiv \sigma(\tau^{n+p}(b))$ for any letter b of the alphabet A .

Define the sequence $(v_k)_{k \geq 0}$ of finite words by $v_0 = u_0 \dots u_{n-1}$ and $v_k = u_{n+(k-1)p} \dots u_{n+kp-1}$ for $k \geq 1$. The word x can be factorized $x = v_0 v_1 v_2 \dots$ and the relations $v_1 \equiv v_2 \equiv v_3 \dots$ hold. This proves that the word x is accepted by the automaton $\mathcal{A}$ iff the ultimately periodic word $v_0 v_1^\omega$ is accepted by $\mathcal{A}$. This can obviously be decided. $\square$

3 A Large Class of Morphic Predicates

The purpose of this section is to give a uniform representation of a large class of morphic predicates: We will show that for $k \geq 0$ and a polynomial $Q(n)$ with non-negative integer values the predicate $P = \{Q(n)k^n \mid n \in \mathbb{N}\}$ is morphic. In particular, the example predicates $\{k^n \mid n \in \mathbb{N}\}$ and $\{n^k \mid n \in \mathbb{N}\}$ for some fixed k in $\mathbb{N}$ of Elgot and Rabin [8] are covered by this.

As a preparation, we shall develop a sufficient condition on sequences $(u_n)_{n \geq 0}$ to define a morphic predicate, involving the notion of a $\mathbb{N}$ -rational sequence. We refer the reader to [12] for a complete introduction and to [2] for a survey, although we recall here the definitions.

Definition 1. A sequence $(u_n)_{n \geq 0}$ of integers is $\mathbb{N}$ -rational if there is a finite graph G allowing multiple edges and sets I and F of vertices such that u_n is the number of paths through G of length n from a vertex of I to a vertex of F . The graph G is said to recognize the sequence $(u_n)_{n \geq 0}$.

An equivalent definition is obtained by considering non-negative matrices. A sequence $(u_n)_{n \geq 0}$ is $\mathbb{N}$ -rational iff there is a matrix M in $\mathbb{N}^{k \times k}$ and two vectors L in $\mathbb{B}^{1 \times k}$ and C in $\mathbb{B}^{k \times 1}$ such that $u_n = LM^n C$. It suffices indeed to consider the adjacency matrix M of the graph and the two characteristic vectors of the sets I and F of vertices. A triple (L, M, C) such that $u_n = LM^n C$ is called a *matrix representation* of the sequence $(u_n)_{n \geq 0}$ and the integer k is called the *dimension* of the representation.

Example 4. The number of successful paths of length n in the graph pictured in Fig. 1 is the Fibonacci number F_n where $F_0 = F_1 = 1$ and $F_{n+2} = F_{n+1} + F_n$. This shows that the sequence $(F_n)_{n \geq 0}$ is $\mathbb{N}$ -rational. The sequence has the following matrix representation of dimension 2:

$$L = (1 \ 0) \quad M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \quad C = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

which can be deduced from the graph of Fig. 1.

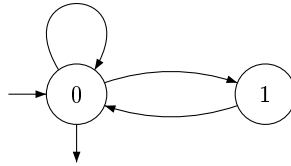


Fig. 1. A graph for the Fibonacci sequence

We are now able to state the main result of this section.

Theorem 2. Let u_n be a sequence of non-negative integers and let $d_n = u_{n+1} - u_n$ be the sequence of differences. If there is some integer l such that $d_n \geq 1$ for any $n \geq l$ and such that the sequence $(d_n)_{n \geq l}$ is $\mathbb{N}$ -rational, then the predicate $P = \{u_n \mid n \in \mathbb{N}\}$ is morphic.

As a first illustration of the Theorem, reconsider the predicate of the Fibonacci numbers F_n : Each difference $d_n = F_{n+1} - F_n$ is equal to F_{n-1} , it obviously satisfies $d_n \geq 1$ for $n \geq 1$, and the sequence $(F_n)_{n \geq 0}$ is $\mathbb{N}$ -rational as shown in Example 4. So the Fibonacci predicate is morphic.

The following corollary provides a large class of morphic predicates:

Corollary 2. *Let Q be a polynomial such that $Q(n)$ is integer for any integer n and let k be a positive integer. Then the predicate $P = \{Q(n)k^n \mid n \in \mathbb{N} \text{ and } Q(n) \geq 0\}$ is morphic.*

The proof of the corollary is entirely based on the following lemma.

Lemma 1. *Let Q be a polynomial with a positive leading coefficient such that $Q(n)$ is integer for any integer n . Let k be a positive integer and let u_n be defined by $u_n = Q(n)k^n$. There is a non-negative integer l such that the sequence $(u_{n+l})_{n \geq 0}$ is $\mathbb{N}$ -rational.*

The following proposition limits the range of morphic predicates; it shows that for example the factorial predicate is not morphic.

Proposition 1. *Let $(u_n)_{n \geq 0}$ be a strictly increasing sequence of integers. If the predicate $\{u_n \mid n \in \mathbb{N}\}$ is morphic, then $u_{n+1} - u_n = O(n^k)$ for some integer k .*

4 Residually Ultimately Periodic Predicates

In this section, we develop a framework which merges the contraction method of Elgot and Rabin with the semigroup approach used for morphic predicates. We capture the reduction to ultimately periodic words as pursued by Elgot and Rabin in the notion of residually ultimately periodic predicate. This is very close to the effectively ultimately periodic reducible predicates introduced by Siefkes [13]. In our general setting, further developed in following section, we can handle the morphic predicates as well as interesting non-morphic ones like the factorial predicate.

Definition 2. *A sequence $(u_n)_{n \geq 0}$ of words over an alphabet A is said to be residually ultimately periodic if for any morphism μ from A^* into a finite semigroup S , the sequence $\mu(u_n)$ is ultimately periodic.*

This property is said to be effective iff for any morphism μ from A^* into a finite semigroup, two integers n and p can be effectively computed such that for any $k \geq n$, $\mu(u_k) = \mu(u_{k+p})$. An infinite word x is called residually ultimately periodic if it can be factorized $x = u_0 u_1 u_2 \dots$ where the sequence $(u_n)_{n \geq 0}$ is effectively residually ultimately periodic. The following proposition is easily verified:

Proposition 2. *If x is residually ultimately periodic, the problem Acc_x is decidable.*

Example 5. The sequence of words $(a^{n!})_{n \geq 0}$ is residually ultimately periodic. This sequence is actually residually ultimately constant. It is indeed well-known that for any element s of a finite semigroup S and any integer n greater than the cardinality of S , $s^{n!}$ is equal to a fixed element, usually denoted s^ω in the literature [1, p. 72].

The sequences of words which are residually ultimately constant have been considerably studied. They are called implicit operations in the literature [1]. A slight variant of the previous example shows that the sequence $(u_n)_{n \geq 0}$ defined by $u_n = 0^{nn!-1}1$ is also residually ultimately constant. Since $(n+1)! - n! - 1$ is equal to $nn! - 1$, the word $u_0u_1u_2 \dots$ is the characteristic word of the factorial predicate $P = \{n! \mid n \in \mathbb{N}\}$. The monadic theory of $\langle \mathbb{N}, <, P \rangle$ where P is the factorial predicate is therefore decidable by the previous proposition.

In the following proposition we connect the sequences obtained by iterating morphisms to the residually ultimately periodic ones.

Proposition 3. *Let τ be a morphism from A^* into itself and let u be a word over A . The sequence $u_n = \tau^n(u)$ is residually ultimately periodic, and this property is effective.*

It follows a morphic word has a factorization whose factors form a residually ultimately periodic sequence.

5 The Predicate Class $\mathcal{K}$

In this section we introduce a large class of residually ultimately periodic predicates defined in terms of strong closure properties (sum, product and exponentiation). This will provide a convenient machinery to find concrete examples of predicates P where the decision problem (Acc_{x_P}) is decidable.

If $(k_n)_{n \geq 0}$ is a strictly increasing sequence of integers, the characteristic word x_P of the predicate $P = \{k_n \mid n \in \mathbb{N}\}$ can be canonically factorized $x = u_0u_1u_2 \dots$ where u_n is the word $0^{k_{n+1}-k_n-1}1$ over the alphabet $\mathbb{B}$. If this “difference sequence” $(u_n)_{n \geq 0}$ of words is residually ultimately periodic and if furthermore this property is effective, then the decision problem (Acc_{x_P}) is decidable.

The following lemma essentially states that it suffices to consider the sequence $u_n = a^{k_{n+1}-k_n}$ over a one-letter alphabet.

Lemma 2. *Let $(u_n)_{n \geq 0}$ be a sequence of words over A and let a be a letter. The sequence $(u_n)_{n \geq 0}$ is residually ultimately periodic iff the sequence $(u_na)_{n \geq 0}$ is residually ultimately periodic. Moreover this property is effective for $(u_n)_{n \geq 0}$ iff it is effective for $(u_na)_{n \geq 0}$.*

If A is the one-letter alphabet $\{a\}$, the semigroup A^* is isomorphic to the set $\mathbb{N}$ of integers by identifying any word a^n with the integer n . Therefore, a sequence $(k_n)_{n \geq 0}$ of integers is said to be residually ultimately periodic iff the sequence (a^{k_n}) is residually ultimately periodic.

Definition 3. *Denote by $\mathcal{K}$ the class of increasing sequences $(k_n)_{n \geq 0}$ of integers such that the sequence $(k_{n+1} - k_n)_{n \geq 0}$ is residually ultimately periodic.*

By the previous lemma and Proposition 2, $\text{MTh}\langle \mathbb{N}, <, P \rangle$ is decidable if the increasing sequence $(k_n)_{n \geq 0}$ which enumerates P belongs to $\mathcal{K}$. The following theorem shows the large extension of $\mathcal{K}$:

Theorem 3. *Any sequence $(k_n)_{n \geq 0}$ such that the sequence $(k_{n+1} - k_n)_{n \geq 0}$ is $\mathbb{N}$ -rational belongs to $\mathcal{K}$. If the sequences $(k_n)_{n \geq 0}$ and $(l_n)_{n \geq 0}$ belong to $\mathcal{K}$, the following sequences also belong to $\mathcal{K}$:*

- (sum and product) $k_n + l_n$ and $k_n l_n$
- (difference) $k_n - l_n$ provided $\lim_{n \rightarrow \infty} ((k_{n+1} - k_n) - (l_{n+1} - l_n)) = \infty$
- (exponentiation) k^{l_n} for a fixed integer k and $k_n^{l_n}$
- (generalized sum and product) $\sum_{i=0}^{l_n} k_i$ and $\prod_{i=0}^{l_n} k_i$.

By Lemma 1, the class $\mathcal{K}$ contains any sequence of the form $k^n Q(n)$ where k is a positive integer and Q is a polynomial such that $Q(n)$ is integer for any integer n . By applying the generalized product to the sequences $k_n = l_n = n$, the sequence $(n!)_{n \geq 0}$ belongs to $\mathcal{K}$.

The closure by differences shows that $\mathcal{K}$ contains any rational sequence $(k_n)_{n \geq 0}$ of integers such that $\lim_{n \rightarrow \infty} (k_{n+1} - k_n) = \infty$. Indeed, any rational sequence of integers is the difference of two $\mathbb{N}$ -rational sequences [12, Cor. II.8.2].

The class $\mathcal{K}$ is also closed by other operations. For instance, it can be proved that if both sequences $(k_n)_{n \geq 0}$ and $(l_n)_{n \geq 0}$ belong to $\mathcal{K}$, then the sequence $(K_n)_{n \geq 0}$ defined by $K_n = \sum_{i+j=n} k_i l_j$ also belongs to $\mathcal{K}$.

Finally, it should be mentioned that $\mathcal{K}$ is not closed under quotient.

Conclusion

We have introduced a large class of unary predicates P over $\mathbb{N}$ such that the corresponding Büchi acceptance problem Acc_{x_P} (and hence the monadic theory $\text{MTh}(\mathbb{N}, <, P)$) is decidable. The class contains all morphic predicates and the examples studied by Elgot and Rabin [8] and Siefkes [13], and it has strong closure properties.

Let us mention some open problems.

Our results do not cover expansions of $\langle \mathbb{N}, < \rangle$ by tuples $(P_1, \dots, P_n)$ of predicates rather than by single predicates. In his dissertation, Hosch [9] has solved the problem for the special case of the predicates $P_i = \{n^{2^i} \mid n \in \mathbb{N}\}$. We do not know whether $\text{MTh}(\mathbb{N}, <, P_1, \dots, P_n)$ is decidable if the P_i are just known to be residually ultimately periodic.

There should be more predicates P for which the acceptance problem Acc_{x_P} and hence the theory $\text{MTh}(\mathbb{N}, <, P)$ is decidable. A possible next step is to consider Sturmian words, a natural generalization of morphic words (see [5]).

Finally, we should recall the intriguing question already asked by Büchi and Landweber in [7] (“Problem 1”): Is there an “interesting” recursive predicate P such that $\text{MTh}(\mathbb{N}, <, P)$ is undecidable? How about P being the prime number predicate?

References

- [1] Jorge Almeida. *Finite Semigroups and Universal Algebra*. World Scientific, 1994.
- [2] Frédérique Bassino, Marie-Pierre Béal, and Dominique Perrin. Length distributions and regular sequences. Technical report, IGM, 2000.
- [3] P. T. Bateman, C. G. Jockusch, and A. R. Woods. Decidability and undecidability of theories of with a predicate for the primes. *J. Symb. Logic*, 58:672–687, 1993.
- [4] Jean Berstel. Axel Thue’s work on repetitions in words. In P. Leroux and C. Reutenauer, editors, *Séries formelles et combinatoire algébrique*, pages 65–80. Publications du LaCIM, Université du Québec à Montréal, 1990.
- [5] Jean Berstel and Patrice Séebold. *Algebraic Combinatorics on Words*, chapter 2, pages 40–96. Cambridge University Press, 2000.
- [6] J. Richard Büchi. On a decision method in the restricted second-order arithmetic. In *Proc. Int. Congress Logic, Methodology and Philosophy of science, Berkeley 1960*, pages 1–14. Stanford University Press, 1962.
- [7] J. Richard Büchi and L. H. Landweber. Definability in the monadic second-order theory of successor. *J. Symb. Logic*, 31:169–181, 1966.
- [8] Calvin C. Elgot and Micheal O. Rabin. Decidability and undecidability of extensions of second (first) order theory of (generalized) successor. *J. Symb. Logic*, 31(2):169–181, 1966.
- [9] F. A. Hosch. *Decision Problems in Büchi’s Sequential Calculus*. Dissertation, University of New Orleans, Louisiana, 1971.
- [10] Arnaud Maes. An automata theoretic decidability proof for the first-order theory of $\langle \mathbb{N}, <, P \rangle$ with morphic predicate P . *Journal of Automata, Languages and Combinatorics*, 4:229–245, 1999.
- [11] C. Michaux and R. Villemaire. Open questions around Büchi and presburger arithmetics. In Wilfrid Hodges et al., editors, *Logic: from foundations to applications. European logic colloquium*, pages 353–383, Oxford, 1996. Clarendon Press.
- [12] Arto Salomaa and Matti Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, New York, 1978.
- [13] D. Siefkes. Decidable extensions of monadic second order successor arithmetic. In J. Doerr and G. Hotz, editors, *Automatentheorie und Formale Sprachen*, pages 441–472, Mannheim, 1970. B.I. Hochschultaschenbücher.
- [14] Wolfgang Thomas. The theory of successor with an extra predicate. *Math. Ann.*, 237(121–132), 1978.
- [15] Wolfgang Thomas. On the bounded monadic theory of well-ordered structures. *J. Symb. Logic*, 45:334–338, 1980.
- [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.

Optical Routing of Uniform Instances in Tori ^{*}

Francesc Comellas¹, Margarida Mitjana², Lata Narayanan³, and
Jaroslav Opatrny³

¹ Departament de Matemàtica Aplicada i Telemàtica, Universitat Politècnica de Catalunya, 08071 Barcelona, Catalonia, Spain, email: comellas@mat.upc.es

² Departament de Matemàtica Aplicada I, Universitat Politècnica de Catalunya 08028 Barcelona, Catalonia, Spain, email: margarida@ma1.upc.es

³ Department of Computer Science, Concordia University, Montreal, Quebec, Canada, H3G 1M8, email: {[@lata](mailto:lata), [@opatrny](mailto:opatrny)}@cs.concordia.ca, FAX (514) 848-2830.

Abstract. We consider the problem of routing *uniform* communication instances in switched optical tori that use the *wavelength division multiplexing* (or WDM) approach. A communication instance is called uniform if it consists exactly of all pairs of nodes in the graph whose distance is equal to one from a specified set $S = \{d_1, d_2, \dots, d_k\}$. We give bounds on the optimal load induced on an edge for any uniform instance in a torus $T_{n \times n}$. When $k = 1$, we prove necessary and sufficient conditions on the value in S relative to n for the wavelength index to be equal to the load. When $k \geq 2$, we show that for any set S , there exists an n_0 , such that for all $n > n_0$, there is an optimal wavelength assignment for the communication instance specified by S on the torus $T_{n \times n}$. We also show an approximation for the wavelength index for any S and n . Finally, we give some results for rectangular tori.

1 Introduction

Optical networks, in which data are transmitted in optical form and where the optical form is maintained for switching, provide transmission rates that are orders of magnitude higher than traditional electronic networks. A single optical fiber can support simultaneous transmission of multiple channels of data, voice and video.

Wavelength-division multiplexing is the most common approach to realize such high-capacity networks [4,5]. A switched optical network using the WDM approach consists of nodes connected by point-to-point fiber-optic links, each of which can support a fixed number of channels or wavelengths. Incoming data streams can be redirected at switches along different outgoing links based on wavelengths. Different data streams can use the same link at the same time as long as they are assigned distinct wavelengths.

Two point x and y that are connected usually have one fiber-optic line for the transmission of signals from x to y and another one for signals from y to x . Thus,

^{*} Research supported in part by NSERC, Canada, and ACI025-1998 Generalitat de Catalunya.

optical networks are generally modeled by *symmetric digraphs*, that is, a directed graph G with vertex set $V(G)$ and edge set $E(G)$ such that the edge $[x, y]$ is in $E(G)$ if and only if the edge $[y, x]$ is also in $E(G)$. In the following, whenever we talk about a graph, we always assume that we consider the associated *symmetric digraph* where any edge between x and y is replaced by two directed edges $[x, y]$ and $[y, x]$.

In a network, a *request* is an ordered pair of nodes (x, y) which corresponds to a message to be sent from x to y . An *instance* I is a collection of requests. Given an instance I in the network, an *optical routing problem* is to determine for each request (x, y) in I a dipath from x to y in the network, and assign it a wavelength, so that any two requests whose dipaths share a link are assigned different wavelengths. Thus, an optical routing problem contains the related tasks of *route assignment* and *wavelength assignment*. A *routing* R for a given instance I is a set of dipaths $\{P(x, y) \mid (x, y) \in I\}$, where $P(x, y)$ is a dipath from x to y in the network. By representing a wavelength by a color, the wavelength assignment can be seen as a coloring problem where one color is assigned to all the edges of a path given by the route assignment. We say that the coloring of a given set of dipaths is *conflict-free* if any two dipaths that share an edge are assigned different colors. Since the cost of an optical switch is proportional to the number of wavelengths it can handle, and the total number of wavelengths that can be handled by a switch is limited, it is important to determine paths and wavelengths so that the total number of required wavelengths is minimized.

Given an instance I in a graph G , and a routing R for it, there are two parameters that are of interest. The *wavelength index* of the routing R , denoted $\mathbf{w}(G, I, R)$, is the *minimum* number of colors needed for a conflict-free assignment of colors to dipaths in the routing R of the instance I in G . The *edge-congestion* or *load* of the routing R for I , denoted by $\pi(G, I, R)$, is the maximum number of dipaths that share the same edge. The parameters $\mathbf{w}(G, I)$, the *optimal wavelength index*, and $\pi(G, I)$, the *optimal load* for the instance I in G are the minimum values over all possible routings for the given instance I in G . It is easy to see that $\mathbf{w}(G, I, R) \geq \pi(G, I, R)$ for every routing R , thus $\mathbf{w}(G, I) \geq \pi(G, I)$. It is known that the inequality can be strict [7]. A general upper bound for $\mathbf{w}(G, I)$ as a function of $\pi(G, I)$ was given in [1]. Determining $\pi(G, I)$ for arbitrary networks and instances is NP-hard [3], though for some specific networks such as trees and rings, and for specific instances, such as the one-to-all instance, the problem can be solved efficiently. Finding $\mathbf{w}(G, I)$ is also NP-hard for arbitrary G and I . In fact, it is known to be NP-hard for specific graphs such as trees and cycles [6]. Approximation algorithms for $\mathbf{w}(G, I)$ have been given for a variety of specific cases; see the survey paper [3].

In view of the NP-hardness of the general case, it is important to characterize the instances for which the wavelength index can be determined efficiently. In this paper, we investigate the wavelength index of *uniform communication instances* on tori. We say that an instance I is uniform if there exists a set of integers $S = \{d_1, d_2, \dots, d_k\}$ such that I consists of all pairs of nodes in G whose distance is equal to d_i for some d_i in the set S . We denote such an instance as I_S . It is

easy to see that the all-to-all instance I_A , consisting of all pairs of nodes of the network, is a special case of a uniform communication instance I_S where $S = \{1, 2, \dots, D_G\}$, and D_G is the diameter of G . Uniform communication instances also occur in certain systolic computations.

A torus of size $m \times n$, denoted $T_{m \times n}$, is a network model having the vertex set $\{(i, j) : 0 \leq i \leq m-1, 0 \leq j \leq n-1\}$, where a vertex (i, j) is connected to vertices $((i+1) \bmod m, j)$, $(i, (j+1) \bmod n)$, $((i-1) \bmod m, j)$, $(i, (j-1) \bmod n)$. In the rest of the paper, all arithmetic operations involving vertices are assumed to be done modulo n or m as appropriate. By mapping vertices into the plane, we can visualize a torus as a graph in which vertices are organized into n rows and m columns, vertex (i, j) belongs to i th column and j th row. Each vertex is connected to the two vertices in the same row and adjacent columns and to the two vertices in the same column and adjacent rows. In the torus, the rows 0 and $n-1$ are adjacent, as are the columns 0 and $m-1$. For a vertex $v = (i, j)$ we call the edges from v to the vertices $(i+1, j)$, $(i-1, j)$, $(i, j+1)$, $(i, j-1)$ its right, left, up, and down edge respectively. The torus is *square* if $m = n$ and is *rectangular* otherwise. We assume without loss of generality that $m \geq n$. The diameter of the torus $T_{m \times n}$ is equal to $\lfloor \frac{m}{2} \rfloor + \lfloor \frac{n}{2} \rfloor$. The *diagonal* D_k of the torus $T_{n \times n}$ consists of the vertices $\{(i, j) \in T_{n \times n} : i + j = k\}$. Similarly, let $D_k^T = \{(i, j) \in T_{n \times n} : i - j = k\}$. The torus network is of interest because it has been used in some highly parallel machines and it is also the underlying virtual network in finite element representation of objects.

The all-to-all communication instance has been considered for several different networks, including a torus [2,11]. For a square torus, $w(T_{n \times n}, I_A) = n^3/8$, which is too large for the present technologies, even for small values of n . Thus, by restricting communications among nodes to pairs of nodes whose distances are in a set S , we can obtain instances that require a substantially smaller number of wavelengths. Some specific uniform instances were previously considered in [9,10] for chordal rings and rings respectively, and general uniform instances were studied in [8] for rings. The problem of wavelength assignment for uniform instances seems to be more difficult than the all-to-all instance for tori, since a uniform instance is an arbitrary subset of the all-to-all instance. At the same time, it is also more complex than the same problem in [8] for rings, as there are many more vertices at distance d from any vertex v in the torus, as well as many more types of dipaths to each destination vertex.

As stated earlier, the task of optical routing involves both path assignment and color assignment to dipaths. In this paper, we always use shortest path routing and unless stated otherwise, the dipath from u to v is the reverse of the dipath from v to u . The colors assigned to a dipath and its reverse are always the same, and hence we speak only about the color assigned to the path between u and v .

To find the wavelength index or an upper bound on it for a uniform instance, we first consider some special cases. The next section presents results for uniform communication instances when S is a singleton set. In particular, we give necessary and sufficient conditions for the wavelength index to be equal to the load,

and an upper bound on the wavelength index is derived in any case. Section 3 considers I_S for $S = \{d_1, d_2\}$. Some sufficient conditions for the wavelength index to be equal to the load are given. The main result in that section is that for any $S = \{d_1, d_2\}$, an optimal wavelength assignment is always possible, provided the torus is large enough. In Section 4 we show that the results from Section 3 can be generalized to get an optimal solution of the general case $S = \{d_1, d_2, \dots, d_k\}$ for a sufficiently large torus and give an approximation in any case. We also give some results for rectangular tori. The last section gives conclusions and some open problems.

Due to space limitations, we mostly give outlines of proofs. Detailed proofs will appear in the full version.

2 Square Tori: Single Path Length

First, we need to derive the value of $\pi(T_{n \times n}, I_S)$, since it gives a lower bound on the wavelength index. The following theorem is stated without proof:

Theorem 1. *Let $T_{n \times n}$ be a square torus and $S = \{d\}$.*

1. *If $d < \lfloor \frac{n}{2} \rfloor$, then $\pi(T_{n \times n}, I_S) = d^2$.*
2. *If $d = \frac{n}{2}$, then $\pi(T_{n \times n}, I_S) = (\frac{n}{2})^2 - \lfloor \frac{n}{4} \rfloor$.*
3. *If $d = D$, the diameter of the torus, then $\pi(T_{n \times n}, I_S) = \frac{D}{4} = \lceil \frac{n}{4} \rceil$ if n is even and $\pi(T_{n \times n}, I_S) = D = n - 1$ if n is odd.*
4. *If $D > d > \lfloor \frac{n}{2} \rfloor$, then $\pi(T_{n \times n}, I_S) = d(2\lfloor \frac{n}{2} \rfloor - d + 1)$.*

We determine when there is a routing and a wavelength assignment that uses exactly the number of colors given by the load of the instance, a lower bound on the wavelength index.

The paths of a routing of shortest paths in a torus can be specified using their *path-types* defined as follows. A dipath has path-type (i, j) where $0 \leq |i|, |j| \leq \lfloor \frac{n}{2} \rfloor$, if it uses $|i|$ horizontal edges and $|j|$ vertical edges. The horizontal edges are *right* edges if i is positive and *left* edges otherwise. Similarly, the vertical edges are *up* edges if j is positive and *down* edges if j is negative. The following theorem provides a necessary condition for the wavelength index to equal the load of a uniform instance.

Theorem 2. *Let $T_{n \times n}$ be a square torus and $S = \{d\}$ where $d < \lfloor \frac{n}{2} \rfloor$. Then $w(T_{n \times n}, I_S) = \pi(T_{n \times n}, I_S) = d^2$ only if d is a factor of $4n^2$.*

Proof: Suppose $w(T_{n \times n}, I_S) = \pi(T_{n \times n}, I_S) = d^2$. Then every color is used on every edge of the network. Fix a color, say c . Let a_i be the number of dipaths of type $(i, d-i)$, $(-i, d-i)$, $(i, -(d-i))$, and $(-i, -(d-i))$, where $0 \leq i \leq d$, that have been assigned the color c . Clearly this accounts for ia_i right or left edges. Since the color c is used on every horizontal edge, we must have $\sum_{i=0}^d ia_i = 2n^2$. Similarly, since c must be used on every vertical edge, $\sum_{i=0}^d (d-i)a_i = 2n^2$. Adding these two equations, we obtain $d \sum_{i=0}^d a_i = 4n^2$, yielding the result. $\square$

Next, we derive sufficient conditions for the wavelength index to equal the load. The key idea to obtain a valid wavelength assignment is as follows: We

define a *band* to be a set of dipaths that are edge-disjoint and can therefore be colored with the same color. A *pattern* is defined as a set of edge-disjoint bands. We always try to find patterns that cover the edge set of the network as much as possible. The wavelength assignment problem can be solved by finding a set of patterns such that their union covers the entire set of dipaths of a given instance. Furthermore, if the set of patterns is such that every pattern contains all edges in the network, and every dipath is contained in exactly one pattern in the set, then the wavelength index equals the load. This idea was also used in [11] to solve the all-to-all instance for tori of even side.

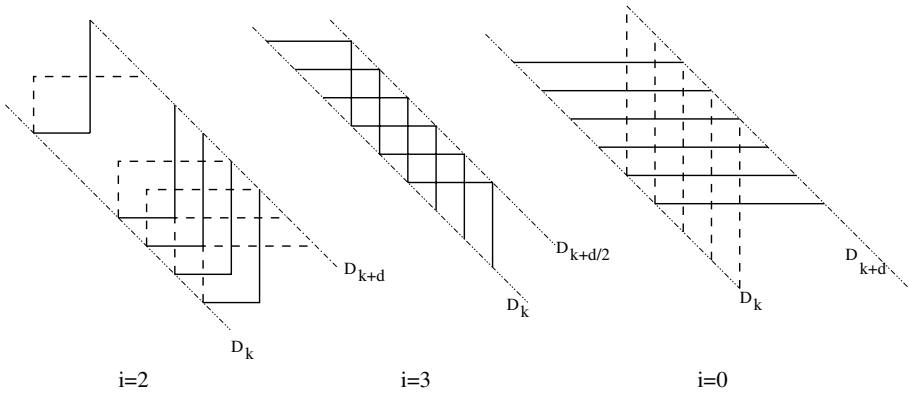


Fig. 1. Bands $A_k(i)$ for $n = 12$ and $d = 6$, and three values of i . Notice that when $i = d/2 = 3$, the band has width $d/2$ and otherwise has width d

We define the *band* $A_k(i)$ (where $i \neq d/2$) to be the set of dipaths from each vertex $(x, y) \in D_k$ to the pair of vertices $(x + i, y + d - i)$ and $(x + d - i, y + i)$ respectively, as well as their reverses. Both of these latter vertices are in D_{k+d} . Furthermore, all edges in between the diagonals D_k and D_{k+d} are covered by the band $A_k(i)$. Notice that the dipaths in the band correspond to 4 different path-types: $(i, d - i)$, $(d - i, i)$, $(-i, -(d - i))$, and $(-(d - i), -i)$, the first two originating in D_k and the last two in D_{k+d} . We call these a set of *companion* path-types. Next, $A_k(\frac{d}{2})$ is defined as the set of dipaths from $(x, y) \in D_k$ to $(x - \frac{d}{2}, y + \frac{d}{2})$. The furthest intermediate vertices form the diagonal $D_{k+\frac{d}{2}}$, and all edges between the diagonals D_k and $D_{k+\frac{d}{2}}$ are covered by the band $A_k(\frac{d}{2})$. The set of companion path-types corresponding to $i = d/2$ contains only two elements, both originating in D_k . See Figure 1 for an example. Similarly, let $B_k(i)$ (where $i \neq \frac{d}{2}$) be the set of dipaths from all vertices $(x, y) \in D_k^T$ to the pair of vertices $(x - i, y + d - i)$ and $(x - d + i, y + i)$ respectively. Note that both of these vertices are in D_{k-d}^T . $B_k(\frac{d}{2})$ is defined analogously to $A_k(\frac{d}{2})$. It is straightforward to see that the width of any band $A_*(i)$ and $B_*(i)$ is d when $i \neq \frac{d}{2}$, and the width of the bands $A_*(\frac{d}{2})$ and $B_*(\frac{d}{2})$ are $d/2$. (We use $A_*(i)$ to denote any band $A_k(i)$ where $0 \leq k \leq n - 1$.) It is not difficult to check that any band defined above is a set of edge-disjoint dipaths.

We define the *pattern* $P_0(i)$, $i = 0 \dots \lceil \frac{d}{2} \rceil - 1$, as the set of bands $A_0(i), A_d(i), A_{2d}(i), \dots, A_{(\lfloor \frac{n}{d} \rfloor - 1)d}(i)$. Similarly, let $P_m(i)$ be the set of bands $\{A_{pd+m}(i) : 0 \leq m \leq d-1 \text{ and } 0 \leq p < \lfloor n/d \rfloor\}$. Finally, let $P_m(\frac{d}{2})$ be the set of bands $\{A_{p\frac{d}{2}+m}(i) : 0 \leq m \leq \frac{d}{2} - 1 \text{ and } 0 \leq p < \lfloor 2n/d \rfloor\}$. The patterns $Q_k(i)$ are defined analogously based on the bands $B_*(i)$.

The next three theorems show that the wavelength index equals the load when the uniform instance given by a single path length d is such that d is a factor of n , the side of the torus.

Theorem 3. *Let $T_{n \times n}$ a the square torus and $S = \{d\}$ where $d < \lfloor \frac{n}{2} \rfloor$. If d is a factor of n then $w(T_{n \times n}, I_S) = \pi(T_{n \times n}, I_S) = d^2$.*

Proof: Since d is a factor of n , the pattern $P_0(i)$ contains all edges in the network, where $0 \leq i \leq \lfloor \frac{d}{2} \rfloor$. The same is true for the patterns $P_1(i), \dots, P_{d-1}(i)$, and the corresponding patterns $Q_*(i)$. Furthermore, it is easy to check that every dipath is included in exactly one of these patterns.

Since each pattern is assigned a color, it suffices to count the number of patterns to determine the wavelengths used. If d is odd, for each value of i , $0 \leq i \leq \lfloor \frac{d}{2} \rfloor$, we need d patterns of type $P_*(i)$. The same is true for patterns of type $Q_*(i)$ except we don't need patterns of this type for $i = 0$ as all dipaths of path-type $(d, 0)$ and $(0, d)$ have already been colored using the P_* patterns. Thus the total number of patterns is $2d\lfloor \frac{d}{2} \rfloor + d = d^2$. If d is even, there are $d\frac{d}{2}$ patterns of type $P_*(i)$ corresponding to $0 \leq i \leq d/2 - 1$, $d(\frac{d}{2} - 1)$ patterns of type $Q_*(i)$ corresponding to $1 \leq i \leq d/2 - 1$, and $d/2$ patterns each corresponding to each of $P_*(\frac{d}{2})$ and $Q_*(\frac{d}{2})$. This adds to a total of $d + 2d(\frac{d}{2} - 1) + d = d^2$ as claimed. $\square$

Theorem 4. *Let $T_{n \times n}$ be a square torus where n is even, and $S = \{\frac{n}{2}\}$. Then $w(T_{n \times n}, I_S) = \pi(T_{n \times n}, I_S) = (\frac{n}{2})^2 - \lfloor \frac{n}{4} \rfloor$.*

Proof: We use the same arguments as in the previous theorems with the exception that the dipaths of path-type $(n/2, 0)$ and $(0, n/2)$ can be colored using $\lceil n/4 \rceil = \lceil d/2 \rceil$ colors, and not d colors as in the previous theorem. $\square$

Theorem 5. *Let $T_{n \times n}$ be the square torus with n^2 vertices and $S = \{d\}$ where $d < \lfloor \frac{n}{2} \rfloor$. If d is even, $d/2$ divides n , and $n \geq d^2/2$, then $w(T_{n \times n}, I_S) = \pi(T_{n \times n}, I_S) = d^2$.*

Proof: If d divides n , then the theorems above show that the wavelength index equals the load. Otherwise, let $n = kd + d/2$. We use two types of patterns to achieve the wavelength assignment. For each $0 \leq i < \frac{d}{2}$, we build a pattern with k bands of type $A_*(i)$ and one of type $A_*(\frac{d}{2})$. We shift this pattern d times, which accounts for bands of type $A_*(i)$ from kd origin diagonals and of type $A_*(d/2)$ from d origin diagonals. Repeating this for all $d/2$ possible values of i , we have bands of type $A_*(i)$ from kd origin diagonals for each i , where $0 \leq i < \frac{d}{2}$ and of type $A_*(\frac{d}{2})$ from $d^2/2$ origin diagonals.

The second type of pattern consists of one band of each type $A_*(i)$ where $0 \leq i < \frac{d}{2}$, and $\frac{n-d^2/2}{d/2}$ bands of type $A_*(\frac{d}{2})$. By shifting this pattern $d/2$ times,

we claim that we can get bands of type $A_*(i)$ where $0 \leq i < \frac{d}{2}$ from $d/2$ origin diagonals, and bands of type $A_*(\frac{d}{2})$ from $n - d^2/2$ origin diagonals.

By using both types of patterns as described, and using a different color for every pattern, we assign colors to all the required dipaths from all origin vertices.

□

Theorem 6. *Let $T_{n \times n}$ be a square torus and $S = \{D\}$ where D is the diameter of the torus. Then $\mathbf{w}(T_{n \times n}, I_S) = \boldsymbol{\pi}(T_{n \times n}, I_S) = \frac{D}{4} = \frac{n}{4}$ if n is even and $\mathbf{w}(T_{n \times n}, I_S) = \boldsymbol{\pi}(T_{n \times n}, I_S) = D = n - 1$ if n is odd.*

Proof: Omitted. □

Next, we give an approximation result for the case of arbitrary $d < \lfloor \frac{n}{2} \rfloor$.

Theorem 7. *Let $T_{n \times n}$ be a square torus and $S = \{d\}$ where $d < \lfloor \frac{n}{2} \rfloor$. Then $\mathbf{w}(T_{n \times n}, I_S) \leq 2\lceil \frac{d}{2} \rceil(d + \frac{n \bmod d}{\lfloor \frac{n}{d} \rfloor})$.*

Proof: Let d be odd. For every i , where $0 \leq i \leq \lfloor \frac{d}{2} \rfloor$, we use a pattern with $\lfloor \frac{n}{d} \rfloor$ bands of type $A_*(i)$. This leaves a “gap” of width $n \bmod d$. We shift this pattern d times, starting the pattern each time with the diagonal where the previous gap started, assigning a new color each time. The remaining origin diagonals, from which dipaths have not yet been assigned colors, are now covered with further patterns using $\frac{n \bmod d}{\lfloor \frac{n}{d} \rfloor}$ more colors. There are $\lceil \frac{d}{2} \rceil$ possible values of i , and symmetry considerations multiply it by 2, giving the result. A similar argument holds when d is even. □

It is not hard to see that the worst case for the above theorem occurs when the gap is of size $d - 1$, i.e., $n \bmod d = d - 1$, yielding the following result:

Theorem 8. *Let $T_{n \times n}$ be a square torus and $S = \{d\}$ where $d < \lfloor \frac{n}{2} \rfloor$. Then $\mathbf{w}(T_{n \times n}, I_S) < 1.5\boldsymbol{\pi}(T_{n \times n}, I_S)$.*

3 Square Tori: Two Path Lengths

When the instance involves more than one path length, we can use Theorem 1 to get a general result about the load of the instance in a square torus.

Theorem 9. *Let $T_{n \times n}$ be a square torus and $S = \{d_1, d_2, \dots, d_k\}$ where $1 \leq d_k < \dots < d_2 < d_1 < \lceil \frac{n}{2} \rceil$. Then the load of $T_{n \times n}$ is $\boldsymbol{\pi}(T_{n \times n}, I_S) = \sum_{i=1}^k d_i^2$.*

Similarly, Theorem 3 can be generalized for S containing more than one path length.

Lemma 1. *Let $T_{n \times n}$ be a square torus and $S = \{d_1, d_2, \dots, d_k\}$ where $1 \leq d_k < \dots < d_2 < d_1 < \lceil \frac{n}{2} \rceil$. If $d_i | n$ for $1 \leq i \leq k$ then $\mathbf{w}(T_{n \times n}, I_S) = \boldsymbol{\pi}(T_{n \times n}, I_S) = \sum_{i=1}^k d_i^2$.*

Proof: If $d_i|n$ we can apply Theorem 3 to each uniform sub-instance consisting of path length d_i and obtain a coloring with d_i^2 colors. Adding up all the contributions, the result follows. $\square$

We will consider here the double-path case $S = \{d_1, d_2\}$ where $1 \leq d_2 < d_1 < \lceil \frac{n}{2} \rceil$. We use some of the results for rings from [8]. First, we show that the wavelength index equals the load when d_1 as well as $d_1 + d_2$ divide n , even when d_2 is not a factor of n .

Lemma 2. *Let $T_{n \times n}$ be a square torus and $S = \{d_1, d_2\}$ where $1 \leq d_2 < d_1 < \lceil \frac{n}{2} \rceil$. If $(d_1 + d_2)|n$ and $d_1|n$ then $w(T_{n \times n}, I_S) = \pi(T_{n \times n}, I_S) = d_1^2 + d_2^2$.*

Proof: We consider the case when d_1 and d_2 are both odd (the other cases are similar). Fix a value of i such that $0 \leq i \leq \lfloor \frac{d_2}{2} \rfloor$. We use a pattern that alternates bands of type $A_*(i)$ of width d_1 and d_2 . This pattern can be shifted $d_1 + d_2$ times, thereby using $d_1 + d_2$ colors to color all dipaths of type $(i, d_1 - i)$ and $(i, d_2 - i)$ as well their companions. We repeat the same procedure for each value of i in the specified range. At this point, all dipaths of length d_2 have been colored. However dipaths of path-type $(i, d_1 - i)$, where $\lfloor \frac{d_2}{2} \rfloor + 1 \leq i \leq \lfloor \frac{d_1}{2} \rfloor$, and their companions have not been assigned colors. Since d_1 divides n , we can now solve these separately, by using patterns that use only bands of width d_1 . Each such pattern can be shifted d_1 times, thus requiring $d_1(\lfloor \frac{d_1}{2} \rfloor - \lfloor \frac{d_2}{2} \rfloor)$ more colors. Thus $w(T_{n \times n}, I_S) = (d_1 + d_2) + 2\lfloor \frac{d_2}{2} \rfloor(d_1 + d_2) + 2(\lfloor \frac{d_1}{2} \rfloor - \lfloor \frac{d_2}{2} \rfloor)(d_1) = d_1^2 + d_2^2$. The factor 2 comes from considering the symmetric bands of type $B_*(i)$. $\square$

Lemma 3. *Let $S = \{d_1, d_2\}$, $1 < d_1 < d_2 < \lfloor \frac{n}{2} \rfloor$, and let $n = a(pd_1) + bd_2$ where $a > b \geq 0$, $a \geq d_2$ and $a - b < pd_1 + d_2$, $1 \leq p \leq \lfloor \frac{d_1}{2} \rfloor$, and pd_1, d_2 and n are mutually co-prime. Then there is an optimal wavelength assignment in $T_{n \times n}$ for all dipaths corresponding to p sets of companion path-types of length d_1 and 1 set of companion path-types of length d_2 .*

Proof: We use similar arguments as in Lemma 2 of [8]. We give only the idea here. We first solve the wavelength assignment problem for one set of companion path-types of length pd_1 and one such set of length d_2 . We use a pattern alternating b bands of width d_2 and pd_1 followed by $a - b$ bands of width d_1 . It is easy to see that this pattern covers the entire edge set. We shift this pattern $i = pd_1 + d_2 - (a - b)$ times, thereby assigning wavelengths to dipaths of length d_1 from ai origin diagonals, and dipaths of length d_2 from bi origin diagonals. As in [8], given the conditions on a and b , we can find a' and b' such that $n = a'd_1 + b'd_2$. We use a second pattern alternating a' bands of width pd_1 and d_2 followed by $b' - a'$ bands of width d_2 , and shift this $j = a - b$ times. Thus we can assign wavelengths to dipaths of length pd_1 from $a'j$ origin diagonals and paths of length d_2 from $b'j$ origin diagonals. It is easy to check that $ai + a'j = bi + b'j = n$, and therefore, all dipaths of length pd_1 and d_2 have been assigned. Finally, each band of width pd_1 is sub-divided into p bands of width d_1 , one for each companion set of path-types, giving the result. $\square$

This brings us to the main theorem of this section:

Theorem 10. *Let $S = \{d_1, d_2\}$ where $1 \leq d_2 < d_1 < \lceil n/2 \rceil$. Then $\mathbf{w}(T_{n \times n}, I_S) = \boldsymbol{\pi}(T_{n \times n}, I_S)$ whenever one of the following holds:*

1. $d_1 | n$ and $d_2 | n$.
2. $(d_1 + d_2) | n$ and $d_1 | n$.
3. $n > (d_1 \lceil \frac{d_1}{d_2-1} \rceil - 2)d_1 \lceil \frac{d_1}{d_2-1} \rceil + (d_1 \lceil \frac{d_1}{d_2-1} \rceil - 1)d_2$ where $d_2 > 1$.

Proof: The first and second statements follow from Lemmas 1 and 2. The key idea for the third statement is that the set of path-types for dipaths of length d_1 and d_2 can always be divided into pairs of subsets such that there are at most $\lceil \frac{d_1}{d_2-1} \rceil$ sets of companion path-types of length d_1 and 1 of length d_2 . We can then apply Lemma 3 to obtain an optimal wavelength assignment. The existence of suitable a and b required by Lemma 3 follows from the arguments in [8]. $\square$

4 The General Case

In this section, we consider the case $S = \{d_1, d_2, \dots, d_k\}$ for $k > 2$. Although the previous two sections dealt mostly with the special cases $S = \{d_1\}$ and $S = \{d_1, d_2\}$, these results can be used to obtain the exact value of $\mathbf{w}(T_{n \times n}, I_S)$ in many instances of the general case.

For example, if $\sum_{i=1}^k d_i$ is a factor of n then $\mathbf{w}(T_{n \times n}, I_S) = \sum_{i=1}^k d_i^2$. We can also obtain a value for $\mathbf{w}(T_{n \times n}, I_S)$ that equals the load, by partitioning S into subsets and applying the results of Section 3. In fact, as in the two path-length case, equality between the wavelength index and the load holds for any arbitrary instance S provided the torus is large enough. If the number of path-lengths k is even, we simply pair the path-lengths, and use Theorem 10 to derive a value of n such that each pair of path-lengths can be solved optimally. If instead k is odd, then we add up two of the path-lengths and reduce to the even case.

This gives the following theorem:

Theorem 11. *Let $S = \{d_1, d_2, \dots, d_k\}$, $1 \leq d_k < \dots < d_2 < d_1 < \lceil n/2 \rceil$. Then there exists an n_0 such that for any $n > n_0$, $\mathbf{w}(R_n, I_S) = \boldsymbol{\pi}(R_n, I_S) = \sum_{i=1}^k d_i^2$.*

An approximation on the wavelength index can be obtained in any case. The proof of the following theorem is based on Theorem 8.

Theorem 12. *Let $S = \{d_1, d_2, \dots, d_k\}$ where $1 \leq d_k < \dots < d_2 < d_1 < \lceil n/2 \rceil$. Then $\mathbf{w}(T_{n \times n}, I_S) \leq 3\boldsymbol{\pi}(T_{n \times n}, I_S)/2$.*

Clearly, the process of decomposition of S into sub-instances of size 1 or 2 and obtaining a solution for I_S by putting together solutions for the sub-instances can be done also in cases when neither one of the two theorems of this section applies to the entire set S .

Next, we consider a rectangular torus $T_{m \times n}$. As mentioned in the introduction, we assume that $m > n$. Many of the results from the previous sections can be generalized to rectangular tori. Using similar arguments as in the case of a square torus, we obtain the following theorem:

Theorem 13. *Let $T_{m \times n}$ be a rectangular torus with $m > n$ and $S = \{d\}$. If $d|m$, $d|n$, and $d \leq n/2$, then $w(T_{m \times n}, I_S) = \pi(T_{m \times n}, I_S)$. Otherwise, if d divides neither n nor m , then $w(T_{m \times n}, I_S) < 2\pi(T_{m \times n}, I_S)$.*

5 Conclusions and Open Problems

In the previous sections we gave exact solutions to find the wavelength index for some uniform instances and derived an approximation of the wavelength index for any uniform instance. The techniques used and results obtained for uniform instances on tori could also be used for deriving results in non-uniform cases.

There remain some open problems: For a single path length $S = \{d\}$, the sufficient and necessary conditions we give for $w(T_{n \times n}, I_S) = \pi(T_{n \times n}, I_S)$ do not match. Can the necessary condition $d|4n^2$ be improved? For the two path length case $S = \{d_1, d_2\}$, is it possible to substantially lower the bound in Theorem 10, part 3, on the dimension of the torus for which $w(T_{n \times n}, I_S) = \pi(T_{n \times n}, I_S)$? Finally, the case of a rectangular tori should be investigated in more detail.

References

1. A Aggarwal, A. Bar-Noy, D. Coppersmith, R. Ramaswami, B. Schieber, and M. Sudan. Efficient routing in optical networks. *JACM*, 46(6):973–1001, 1996.
2. B. Beauquier. All-to-all communication for some wavelength-routed all-optical networks. Technical report, INRIA Sophia-Antipolis, 1998.
3. B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Perennes, and U. Vaccaro. Graph problems arising from wavelength-routing in all-optical networks. In *Proceedings of the 2nd Workshop on Optics and Computer Science (WOCS), part of IPPS*, April 1997.
4. C. Bracket. Dense wavelength division multiplexing networks: Principles and applications. *IEEE J. Selected Areas in Communications*, 8:948–964, 1990.
5. N.K. Cheung, K. Nosu, and G. Winzer. An introduction to the special issue on dense WDM networks. *IEEE J. Selected Areas in Communications*, 8:945–947, 1990.
6. T. Erlach and K. Jansen. Scheduling of virtual connections in fast networks. In *Proceedings of the 4-th Workshop on Parallel Systems and Algorithms*, pages 13–32, 1996.
7. M. Mihail, C. Kaklamanis, and S. Rao. Efficient access to optical bandwidth. In *FOCS*, pages 548–557, 1995.
8. L. Narayanan and J. Opatrny. Wavelength routing of uniform instances in optical rings. In *Proceedings of ARACNE 2000*, 2000. To appear.
9. L. Narayanan, J. Opatrny, and D. Sotteau. All-to-all optical routing in chordal rings of degree four. In *Proceedings of the Symposium on Discrete Algorithms*, pages 695–703, 1999.
10. J. Opatrny. Uniform multi-hop all-to-all optical routings in rings. In *Proceedings of LATIN'2000, LNCS 1776*, pages 237–246, 2000.
11. H. Schroder, O. Sykora, and I. Vrto. Optical all-to-all communication for some product graphs. In *Theory and Practice of Informatics, Seminar on Current Trends in Theory and Practice of Informatics, LNCS*, volume 24, 1997.

Factorizing Codes and Schützenberger Conjectures [★]

Clelia De Felice

Dipartimento di Informatica ed Applicazioni
Università di Salerno
84081 Baronissi (SA), Italy.
fax number: + 39 – (0)89965272
defelice@dia.unisa.it

Abstract. In this paper we mainly deal with *factorizing codes* C over A , i.e., codes verifying the famous still open *factorization conjecture* formulated by Schützenberger. Suppose $A = \{a, b\}$ and denote a^n the power of a in C . We show how we can construct C starting with factorizing codes C' with $a^{n'} \in C'$ and $n' < n$, under the hypothesis that all words $a^i w a^j$ in C , with $w \in bA^*b \cup \{b\}$, satisfy $i, j < n$. The operation involved, already introduced in [1], is also used to show that all maximal codes $C = P(A - 1)S + 1$ with $P, S \in \mathbf{Z}\langle A \rangle$ and P or S in $\mathbf{Z}\langle a \rangle$ can be constructed by means of this operation starting from prefix and suffix codes. Inspired by another early Schützenberger conjecture, we propose here an open problem related to the results obtained and to the operation introduced in [1] and considered in this paper.

1 Introduction

The notion of code appears in a natural way in Computer Science and in the theory of Information when we need to adapt original messages to a transmission channel (consider, for instance, the binary representation of numbers and instructions in computers or the Morse code). The algebraic approach initiated by Schützenberger in [23] - a *variable-length code* is the base of a free submonoid of a free monoid - made the theory of codes a topic of interest also from a mathematical point of view with connections with other fields and problems. At the beginning of the theory several conjectures on the structure of codes were proposed by Schützenberger, making their description and their construction by means of procedures one of the goals of the theory. In this paper we present some results and an open problem which go in this direction.

An early Schützenberger conjecture asked whether each finite *maximal code* (i.e., a maximal object in the class of finite codes for the order of set inclusion) could be obtained by means of a simple operation, called *composition* of codes,

[★] Partially supported by MURST Project “*Unconventional Computational Models: Syntactic and Combinatorial Methods*” - “*Modelli di calcolo innovativi: Metodi sintattici e combinatori*”.

starting from prefix or suffix codes. This conjecture was false: a first counterexample was given by Césari in [9] and subsequently simpler codes were constructed by Boë and Vincent in [4] and [24] respectively. Another famous conjecture, which is still open and is known as the *factorization conjecture*, states that given a finite maximal code C , there would be finite subsets P, S of A^* such that $\underline{C} - 1 = \underline{P}(\underline{A} - 1)\underline{S}$, with $\underline{X}$ denoting the characteristic polynomial of X [2,3]. Each code C which satisfies the above equality is finite, maximal, and is called a *factorizing code*.

In this paper we give evidence of the existence of an algorithm for constructing factorizing codes. Precisely, let C be a factorizing code over $\{a, b\}$ and let a^n be the power of a in C . In Proposition 1, we show how we can construct C starting with factorizing codes C' with $a^{n'} \in C'$ and $n' < n$, under the hypothesis that all words $a^i w a^j$ in C , with $w \in bA^*b \cup \{b\}$, satisfy $i, j < n$. Observe that for constructing all factorizing codes, what is missing is a transformation which allows us to go from a factorizing code C with $a^n \in C$ to a (factorizing) code, denoted $C^{(\bmod n)}$, which is roughly obtained from C by reducing the exponents of the a 's in C modulo n . The operation involved in Proposition 1, already introduced in [1] where it was called *substitution*, is also used to show that all maximal codes $C = P(A - 1)S + 1$ with $P, S \in \mathbf{Z}\langle A \rangle$ and P or S in $\mathbf{Z}\langle a \rangle$ can be constructed by means of this operation starting from prefix and suffix codes (see Section 3).

Then, a natural question which arises is which factorizing codes can be obtained by substitution of prefix and suffix codes. We do not yet know of any examples which give a negative answer to the following problem:

Problem 1. Can each factorizing code be obtained by substitution of prefix and suffix codes?

If both the factorization conjecture and Problem 1 could be solved positively, the first Schützenberger conjecture could be formulated differently as follows: “Each finite maximal code can be obtained by substitution of prefix and suffix codes” (and should have a positive answer). It is worthy of note that all the counterexamples to the above-mentioned first Schützenberger conjecture can be obtained by substitution starting from prefix or suffix codes and in order to show this we state that the same result holds for the class of factorizing codes $C = P(A - 1)(1 + w) + 1$, $w \in A^*$ (Section 4).

All the proofs which are not contained in this paper can be found in a manuscript which is available upon request to the author [14].

2 Basics

2.1 Codes and Polynomials

Let A^* be the *free monoid* generated by a finite alphabet A and $A^+ = A^* \setminus 1$ where 1 is the empty word. For a word $w \in A^*$, we also denote the *length* of w by $|w|$. When referred to a set X , the notation $|X|$ means the cardinality of X .

A code C is a subset of A^* such that, for any $c_1, \dots, c_h, c'_1, \dots, c'_k \in C$, we have:

$$c_1 \cdots c_h = c'_1 \cdots c'_k \Rightarrow h = k; \quad \forall i \in \{1, \dots, h\} \quad c_i = c'_i.$$

Any set $C \subseteq A^+$ such that $C \cap CA^+ = \emptyset$ is a *prefix* code. C is a *suffix* code if $C \cap A^+C = \emptyset$ and C is a *biprefix* code when C is both a suffix and a prefix code. A code C is a *maximal* code over A if for each code C' over A such that $C \subseteq C'$ we have $C = C'$. A code C is *synchronous* if C has degree 1, otherwise C is called *asynchronous* (see [2] for the definition of the degree of a code).

Denote $\mathbf{Z}\langle A \rangle$ (respectively $\mathbf{N}\langle A \rangle$) the semiring of the *polynomials* with non-commutative variables in A and integer (respectively nonnegative integer) coefficients. Each finite subset X of A^* will be identified with its *characteristic polynomial*: $\underline{X} = \sum_{x \in X} x$. Henceforth we will use a capital letter to refer to a set and to its characteristic polynomial. For a polynomial P and a word $w \in A^*$, (P, w) denotes the coefficient of w in P and $P \geq 0$ means $P \in \mathbf{N}\langle A \rangle$. For a subset H of $\mathbf{N}$, we also denote $a^H = \sum_{h \in H} a^h$.

One of the most difficult problems which is still open in the theory of codes was proposed by Schützenberger and is known as the *factorization conjecture*:

Conjecture 1. [2,3] Given a finite maximal code C , there are finite subsets P, S of A^* such that

$$C - 1 = P(A - 1)S \quad (1)$$

Each code C verifying the previous conjecture is finite, maximal and is called a *factorizing code*. (P, S) is known as a *factorizing pair* for C . The conjecture is still open and weaker forms of it have been given [20]. The first examples of families of factorizing codes can be found in [5] and the result which is closest to a solution of the conjecture, partially reported in Theorem 1, was obtained by Reutenauer [3,22]: he proved that (1) holds for each finite maximal code C and with $P, S \in \mathbf{Z}\langle A \rangle$.

Theorem 1. [22] Let C be such that $C \in \mathbf{N}\langle A \rangle$, $(C, 1) = 0$ and let P, S be such that $P, S \in \mathbf{Z}\langle A \rangle$, $C = P(A - 1)S + 1$. Then, C is a finite maximal code. Furthermore, if $P, S \in \mathbf{N}\langle A \rangle$, then P, S have coefficients 0, 1. Conversely, let C be a finite maximal code. Then, there exist P, S such that $P, S \in \mathbf{Z}\langle A \rangle$ and $C = P(A - 1)S + 1$.

Given a factorizing code C with $a^n \in C$ and a factorizing pair (P, S) for C , it is easy to see that $P \cap a^* = a^I$, $S \cap a^* = a^J$ where $a^I a^J = (a^n - 1)/(a - 1)$. Such pairs (I, J) were considered by Krasner and Ranulac in [17] and called *Krasner factorizations* here. Given a Krasner factorization (I, J) with $(I, J) \neq (0, 0)$, it is easy to see that there exists k with $k \in \mathbf{N}$, $k < n$, $k|n$, such that either $I = I_1 + \{0, \dots, (n/k) - 1\}k$, $I_1 \subseteq \mathbf{N}$, $a^{I_1} a^J = (a^k - 1)/(a - 1)$ or $J = J_1 + \{0, \dots, (n/k) - 1\}k$, $J_1 \subseteq \mathbf{N}$, $a^I a^{J_1} = (a^k - 1)/(a - 1)$ [18].

2.2 Operations on Codes

One way to simplify the problem of the description of the structure of the (finite maximal) codes is to use operations which allow us to construct codes starting with “simpler” ones. One of these operations is the *composition* of codes. A code $C \subseteq A^*$ *decomposes* over a code D if $C \subseteq D^*$. Suppose that the latter relation is satisfied with D of minimal cardinality and let B be an alphabet with the same cardinality as D . Thus, a code $Y \subseteq B^*$ exists such that $\varphi(Y) = C$, $\varphi : B^* \rightarrow D^*$ being an isomorphism. We also say that C is obtained by *composition* of Y and D . The class of factorizing codes was shown to be closed under composition in [5] and specific relations exist between factorizing pairs for Y and D and a factorizing pair for C [5]. The same class of factorizing codes is also closed under another operation which was considered in [1] for the first time. Let us briefly recall it.

Given factorizing codes $C = P(A-1)S+1$, $C' = P'(A-1)S+1$ and $w \in C$, by using Theorem 1, it is clear that $C'' = (P+wP')(A-1)S+1 = C-w+wC'$ is again a factorizing code which is called a *substitution* of C and C' by means of w . More generally, the result of a finite number of applications of such an operation, or of a dual version of it (working on $C = P(A-1)S+1$, $C' = P(A-1)S'+1$ and $w \in C$) once again will be called “substitution”. If C decomposes over D and particular hypotheses on factorizing pairs for C and D are satisfied, then C can be obtained by substitution of D . We do not know whether this result would still hold without these hypotheses, i.e., if any code C which decomposes over D can be obtained by substitution of D .

As in the composition of codes, substitution could also be defined by starting with two finite maximal codes $C = P(A-1)S+1$, $C' = P'(A-1)S+1$ and $w \in C$, where P, P', S are now polynomials in $\mathbf{Z}\langle A \rangle$. Once again by using Theorem 1, we have that $C'' = P(A-1)S+wP'(A-1)S+1 = C-w+wC'$ is the characteristic polynomial of a finite maximal code.

3 Main Results

In this section we suppose $A = \{a, b\}$ and we give two results related to Problem 1. In particular, given a factorizing code C which satisfies a particular hypothesis, Proposition 1 shows how we can obtain each factorizing pair (P, S) for $C = P(A-1)S+1$, starting from factorizing pairs (P', S') for codes with a smaller power of a inside them.

Proposition 1. *Let C be a factorizing code with $a^n \in C$, $n > 1$. Suppose that for each $a^i w a^j \in C$, with $w \in bA^*b \cup \{b\}$, we have $i, j < n$. Then, for each factorizing pair (P, S) for $C = P(A-1)S+1$, there exist k with $k \in \mathbf{N}$, $k|n$, $k < n$ and there exist factorizing codes $C^{(h)}$ with $a^k \in C^{(h)}$, $h \in \{0, \dots, (n/k)-1\}$, such that we have*

$$\begin{aligned} & - \text{ either } C^{(h)} = P^{(h)}(A-1)S+1, P = \sum_{h=0}^{(n/k)-1} a^{hk} P^{(h)} \text{ and} \\ & C = \sum_{h=0}^{(\frac{n}{k}-1)} a^{hk} (C^{(h)} - 1) + 1 \end{aligned}$$

$$\begin{aligned} & - \text{ or } C^{(h)} = P(A-1)S^{(h)} + 1, S = \sum_{h=0}^{(n/k)-1} S^{(h)} a^{hk} \text{ and} \\ & C = \sum_{h=0}^{(\frac{n}{k}-1)} (C^{(h)} - 1) a^{hk} + 1 \end{aligned}$$

the first or the second condition being satisfied according whether $P \cap a^* = a^{I_1 + \{0, \dots, (n/k)-1\}k}$ or $S \cap a^* = a^{J_1 + \{0, \dots, (n/k)-1\}k}$.

Remark 1. Let n, k be positive integers with $k|n$, $n/k \geq 2$, and let $C^{(h)}$ be a factorizing code with $a^k \in C^{(h)}$, for $h \in \{0, \dots, (n/k) - 1\}$. Thus, $C = \sum_{h=0}^{(n/k)-1} a^{hk}(C^{(h)} - 1) + 1$ is obtained by substitution of codes $C^{(h)}$ and so C is a factorizing code. Indeed, we have $C = \sum_{h=0}^{(n/k)-1} a^{hk}(C^{(h)} - 1) + 1 = (C^{(0)} - 1) + a^k(\sum_{h=1}^{(n/k)-1} a^{(h-1)k}(C^{(h)} - 1)) + 1$. So, if $n/k = 2$ we see that C is obtained by substitution of $C^{(0)}$ and $C^{(1)}$ and the conclusion follows by induction over n/k .

We now illustrate the above statement through an example.

Example 1. Consider the factorizing code $C = (1 + ba + aA + aA^2a)(A-1)(1 + A + b^2)$ (Césari, in [2,9]). The Krasner pair (I, J) associated with C is $I = \{0, 2, 4\}$, $J = \{0, 1\}$, $n = 6$ and I has period $k = 2$. Furthermore, C satisfies the hypothesis of Proposition 1 and, according to this proposition, we have $C = \sum_{h=0}^2 a^{2h}(C^{(h)} - 1) + 1$ with $C^{(0)} - 1 = (1 + ab + ba + aba^2 + ab^2a)(A-1)(1 + A + b^2)$, $C^{(1)} - 1 = (1 + ba)(A-1)(1 + A + b^2)$, $C^{(2)} - 1 = (A-1)(1 + A + b^2)$. In addition, observe that C can be obtained by substitution starting with suffix codes (see Problem 1). Indeed, C is obtained by substitution using $C^{(0)}$, $C^{(1)}$ and $C^{(2)}$. Now, $C^{(2)} = \{ba, ab^2, b^3, a^2, ab\}$ is a suffix code and $C^{(1)}$ is obtained by substitution of $C^{(2)}$ and $C^{(2)}$ by means of ba . Finally, since $a^2, ab \in C^{(1)}$, then $C' = (1 + ba + a^2)(A-1)(1 + A + b^2) + 1 = (1 + ba)(A-1)(1 + A + b^2) + a^2(A-1)(1 + A + b^2) + 1$ is obtained by substitution of $C^{(1)}$ and $C^{(2)}$ by means of $a^2 \in C^{(1)}$ whereas $C^{(0)} = (1 + ba + ab(1 + a^2 + ba))(A-1)(1 + A + b^2) + 1 = (1 + ba)(A-1)(1 + A + b^2) + ab(1 + a^2 + ba)(A-1)(1 + A + b^2) + 1$ is obtained by substitution of $C^{(1)}$ and C' by means of $ab \in C^{(1)}$.

We now recall the definition of a family of codes for which we see that Problem 1 has a positive answer. An m -code C is a finite maximal code with at most m occurrences of b 's in each word of C . We know that 1-, 2- and 3 - codes are factorizing (see [15,12,21]). The same result holds for p -codes C with $b^p \in C$ and $p = 4$ or p a prime number [25]. 2-codes (and some 3 - codes) belong to the class of finite maximal codes $C = P(A-1)S + 1$, with $P, S \in \mathbf{Z}\langle A \rangle$ and $P \in \mathbf{Z}\langle a \rangle$ or $S \in \mathbf{Z}\langle a \rangle$. The structure of the latter codes, shown to be factorizing in [12], is related to the so-called *Hajós factorizations* of a cyclic group $\mathbf{Z}_n$ (see [11,13,16, 18] for more details).

Proposition 2. *Let C be a finite maximal code such that $C = P(A-1)S + 1$, with $P, S \in \mathbf{Z}\langle A \rangle$ and $P \in \mathbf{Z}\langle a \rangle$ or $S \in \mathbf{Z}\langle a \rangle$. Then C can be obtained by substitution of prefix and suffix codes.*

Example 2. Proof of Proposition 2 gives a procedure for finding the prefix or suffix codes in the substitution. We now illustrate this procedure through this example. Consider the synchronous indecomposable 3-code $C = 1 + a^{\{0,2\}}(A - 1)(a^{\{0,1\}} + a^{\{2,3\}}b + a^{\{0,3,4\}}ba^2b)$ [6]. We have $C - 1 = (C' - 1) + (C'' - 1)z = (a^{\{0,2\}}(A - 1)a^{\{0,1\}}) + (a^{\{0,2\}}(A - 1)(a^{\{0,1\}} + a^{\{0,3,4\}}b))a^2b$. Evidently $a^2b \in C'$ and C can be obtained by substitution of C' and C'' by means of a^2b . Furthermore, C', C'' still belong to the same class of codes described in Proposition 2 but with a smaller cardinality than C . However, $C' = 1 + a^{\{0,2\}}(A - 1)a^{\{0,1\}}$ can be obtained by substitution of the suffix code $G = (A - 1)a^{\{0,1\}} + 1 = \{b, ba, a^2\}$. For C'' a second application of Proposition 2 yields $C'' - 1 = (D - 1) + (D' - 1)y = (D - 1) + (C' - 1)y = (a^{\{0,2\}}(A - 1)(a^{\{0,1\}} + b)) + (a^{\{0,2\}}(A - 1)a^{\{0,1\}})a^3b$. We see that $a^3b \in D$. Furthermore, D can be obtained by substitution of the prefix code $A^2 = (A - 1)(a^{\{0,1\}} + b) + 1$.

4 Constructions of Codes and Schützenberger Conjectures

As we already said, Problem 1 was inspired by an old conjecture formulated by Schützenberger. Counterexamples to this conjecture were obtained by different constructions. In this section we show how the class of factorizing codes produced using the substitution operation strictly contains all those codes. We have already done it for the Césari code (see Example 1).

Let us consider the class of codes $\mathcal{F}$ constructed by Boë in [4] and defined as follows. Let $X = P(A - 1) + 1$ and $Y = Q(A - 1) + 1$ be two prefix codes such that $X \cap Y \neq \emptyset$. As observed in [4], for any finite set R such that $R \subseteq (X \cap Y)^*$ and satisfying $v \in R$ whenever $uv \in R$ and $u \in (X \cap Y)^*$, the relations $Z = (X \cap Y - 1)R$, $E = P(A - 1)R + 1$ and $F = Q(A - 1)R + 1$ define three codes Z, E, F . Let w be a word of maximal length in E , a code C belongs to $\mathcal{F}$ if we have $C = (P + wQ)(A - 1)R + 1$. Obviously, C can be obtained by substitution of the codes E and F whereas in Example 2, we give an example of a code C which can be obtained by substitution of codes and which does not belong to $\mathcal{F}$. As mentioned in [4], the hypothesis on the maximality of the length of w is unnecessary provided that C is a code, as always happens when $w \in E$, thanks to Theorem 1. As recalled in Section 1, under particular hypotheses, a code $C \in \mathcal{F}$ is indecomposable and neither a prefix code nor a suffix code.

Example 2 (continued). Consider the 3-code $C = 1 + a^{\{0,2\}}(A - 1)(a^{\{0,1\}} + a^{\{2,3\}}b + a^{\{0,3,4\}}ba^2b)$. As we have already seen, C can be obtained by substitution of prefix and suffix codes. As a consequence of a result shown in [7], C is synchronous, indecomposable and it has a unique factorization: the pair $(a^{\{0,2\}}, a^{\{0,1\}} + a^{\{2,3\}}b + a^{\{0,3,4\}}ba^2b)$ (see Theorems 4.1, 4.14 and Example 4.2 in [7]). Let us prove that C cannot be obtained by using Boë's construction. By contradiction, suppose that $C \in \mathcal{F}$. With the same notations as above, since C has a unique factorization, we should have $a^{\{0,2\}} = P + wQ$ and $a^{\{0,1\}} + a^{\{2,3\}}b + a^{\{0,3,4\}}ba^2b = R$. This is impossible however, since P, Q being prefix-closed, we should also have $P = Q = 1$, i.e., $X = Y = A$, and R should be

suffix-closed, which it is not. Nor can we obtain C by using Boë's dual construction, i.e., we cannot have $a^{\{0,2\}} = R$, $a^{\{0,1\}} + a^{\{2,3\}}b + a^{\{0,3,4\}}ba^2b = P + Qw$, with P, Q, R such that $(A-1)P+1$, $(A-1)Q+1$ are suffix codes, $R(A-1)P+1$, $R(A-1)Q+1$ are codes, $w \in R(A-1)P+1$ (P being suffix-closed, we should have $P = 1 + a$. But now $a^{\{2,3\}}b + a^{\{0,3,4\}}ba^2b$ cannot be equal to Qw with Q suffix-closed).

Other constructions of codes were given in [24] producing, under particular hypotheses, codes which are indecomposable over a prefix or a suffix code. These constructions look a bit like the construction of the finite biprefix codes given in [8] or of the asynchronous prefix codes, given in [19]. Once again, we will see that the class of codes obtained by substitution of prefix and suffix codes strictly contains codes constructed in [24]. This is the result of three facts.

Firstly, a code C obtained by using Vincent's construction has the form $T_d = (P_X - yP_D + GyP_D)(A-1)(1+y) + 1$ or it has the form $I_d = (T_g - 1 + (G' - 1)y^2(D' - 1))(1 + y^2) + 1 = (P_g + (G' - 1)y^2(1+y)P_D)(A-1)(1+y^2) + 1$, where $P_X - yP_D + GyP_D$ and $P_g + (G' - 1)y^2(1+y)P_D$ are polynomials with coefficients 0, 1, $y \in A^+$ and other particular hypotheses are imposed on the sets appearing in these expressions. Thus, in both cases, a code C obtained by using Vincent's construction is a factorizing code with the form $C = P(A-1)S + 1$ and $|S| = 2$.

On the other hand, as we will state in Proposition 4, each code C with the form $C = P(A-1)(1+w)$, $w \in A^+$, can be obtained by substitution of prefix and suffix codes.

Finally, it is obvious that factorizing codes C exist which can be obtained by substitution of prefix and suffix codes and such that $C = P(A-1)S + 1$ with $|P| > 2$ and $|S| > 2$. In order to show this, let us consider the class of codes C introduced in [21]. Each such code C can be written as $C = a^I(A-1)a^J + 1$, with (I, J) being a Krasner factorization. It is clear that we can apply Proposition 1 to C and we obtain codes C' which belong to the same class but with a smaller power of a inside them. So, the latter proposition can be recursively applied until we get prefix or suffix codes and each C can be obtained by substitution of prefix and suffix codes. As observed in [6], these codes have the unique factorization (a^I, a^J) and we can choose I, J with $|I| > 2$, $|J| > 2$.

In order to show Proposition 4, we recall the description of the structure of the finite subsets P of A^* such that $P(A-1)(1+w) + 1 \geq 0$, $w \in A^+$ as described in [10].

Proposition 3. [10] *Let $w \in A^+$ and let P_0 be the set of the proper prefixes of w . We have $P(A-1)(1+w) + 1 \geq 0$ if and only if $1 \in P$ and for each $z \in P \setminus PA$ we have $zP_0 \subseteq P$ and $zw \notin P$. Furthermore, if $z \neq 1$, there exist $a \in A$ and $p \in P$ such that $z = paw$ and $paP_0 \cap P = \emptyset$.*

Proposition 4. *Let C be a factorizing code such that $C = P(A-1)(1+w) + 1$ with $w \in A^+$ and P a finite subset of A^* . Then C can be obtained by substitution of prefix and suffix codes.*

Proof. Let P be a finite subset of A^* such that $C = P(A-1)(1+w) + 1 \geq 0$. We will prove our conclusion by induction over $|P|$.

If P is prefix-closed, then we have our conclusion since $C_0 = P(A-1) + 1 \geq 0$, thus C_0 is a prefix code, $w \in C_0$ ($w \in PA \setminus P$ since $P_0 \subseteq P$ and $w \notin P$) and C is obtained by substitution of C_0 by means of w .

If P is not prefix-closed, then P_0 is a proper subset of P and a word y exists in $P \setminus PA$. Thus, $y = paw$, y satisfies all the conditions in Proposition 3 and we can choose y with p of maximal length in P . Since $yP_0 \subseteq P$ we can write P as $P = P_1 + pawP_0 + pawP_2 + \dots + pawP_g$ with $P_0, P_2, \dots, P_g$ prefix-closed and $P_1 \cap (P \setminus P_1)A = \emptyset$. Observe that with our notations, we have $p \in P_1$.

We first prove that $P_1(A-1)(1+w) + 1 \geq 0$ by showing that all the conditions in Proposition 3 are satisfied for P_1 .

We know that, as $C = P(A-1)(1+w) + 1 \geq 0$, P satisfies all the conditions in Proposition 3. Obviously, $P_0 \subseteq P_1$ and $w \notin P_1$ (since $w \notin P$). Furthermore, let $z \in P_1 \setminus P_1A$, $z \neq 1$. We have $z \in P \setminus PA$, $zw \notin P_1$ (since $zw \notin P$) and there exist $p' \in P$, $a' \in A$ such that $z = p'a'w$. We also have $p'a'P_0 \cap P_1 = \emptyset$ (since $p'a'P_0 \cap P = \emptyset$) and $p'a'wP_0 \subseteq P$. Thus, using Proposition 3, in order to show that $P_1(A-1)(1+w) + 1 \geq 0$, we only have to prove that $p' \in P_1$ and $p'a'wP_0 \subseteq P_1$.

Obviously we have $p' \in P_1$ since $p' \in P \setminus P_1$ implies $p' = paww_1$ with $w_1 \in P_j$, $j \in \{0, 2, \dots, g\}$, and $|p'| > |p|$ with $p'a'w \in P \setminus PA$, which contradicts the definition of p . We also have $p'a'wP_0 \subseteq P_1$ since otherwise, there exists $p_k \in P_0$ such that $p'a'wp_k = paww_1$ with $w_1 \in P_j$. We cannot have $|p| > |p'|$ since $|p| > |p'|$ and $p \in P$ imply, in virtue of Proposition 3 applied to $z = p'a'w$, $|p| \geq |p'a'w|$ with $|p_k| < |w|$: a contradiction with $p'a'wp_k = paww_1$. Furthermore, we cannot have $|p'| = |p|$ since otherwise, $z = y \notin P_1$. Thus, $|p'| > |p|$ with $p'a'w \in P \setminus PA$, as already seen in contradiction with the definition of p .

Thus, P_1 is such that $C_0 = P_1(A-1)(1+w) + 1 \geq 0$ and $|P_1| < |P|$: by using the induction hypothesis, C_0 is a code obtained by substitution using prefix and suffix codes. We also have $y = paw \in P_1Aw \setminus (P_1w \cup P_1)$, thus $paw \in C_0$. Furthermore, by using once again Proposition 3, $P' = P_0 + P_2 + \dots + P_g$ is such that $C'_0 = P'(A-1)(1+w) + 1 \geq 0$. Indeed, the unique word $z \in P' \setminus P'A$ is the empty word, $P_0 \subseteq P'$ and $w \notin P'$ (if w were in P' , then $paww \in P$ with $paw \in P \setminus PA$: a contradiction with Proposition 3 applied with P and $y = paw$). Since P' is prefix-closed, we have already proved that C'_0 is a code obtained by substitution using prefix and suffix codes.

Finally, $C = (P_1 + pawP')(A-1)(1+w) + 1$ and C can be obtained by substitution of C_0 and C'_0 by means of $paw \in C_0$.

□

Acknowledgments. The author wishes to thank V. Bruyère and A. Restivo for helpful suggestions.

References

1. M. Anselmo, A Non-Ambiguous Languages Factorization Problem, in: *Preproc. DLT'99 Aachener Informatik-Berichte* **99-5** (1999) 103–115.

2. J. Berstel, D. Perrin, *Theory of Codes*, Academic Press, New York (1985).
3. J. Berstel, C. Reutenauer, *Rational series and their languages*, EATCS Monogr. Theoret. Comput. Sci. **12**, Springer Verlag (1988).
4. J. M. Boë, Une famille remarquable de codes indécomposables, *Lecture Notes in Computer Science* **62** (1978) 105–112.
5. J. M. Boë, Sur les codes factorisants, in: “*Théorie des codes*” (D. Perrin ed.), LITP (1979) 1–8.
6. V. Bruyère, C. De Felice, Synchronization and decomposability for a family of codes, *Intern. Journ. of Algebra and Comput.* **4** (1992) 367–393.
7. V. Bruyère, C. De Felice, Synchronization and decomposability for a family of codes: Part 2 *Discrete Math.* **140** (1995) 47–77.
8. Y. Césari, Sur un algorithme donnant les codes bipréfixes finis, *Math. Syst. Theory* **6** (1972) 221–225.
9. Y. Césari, Sur l’application du théorème de Suschkevitch à l’étude des codes rationnels complets, *Lecture Notes in Computer Science* (1974) 342–350.
10. C. De Felice, Construction de codes factorisants, *Theoret. Comput. Sci.* **36** (1985) 99–108.
11. C. De Felice, Construction of a family of finite maximal codes, *Theoret. Comput. Sci.* **63** (1989) 157–184.
12. C. De Felice, A partial result about the factorization conjecture for finite variable-length codes, *Discrete Math.* **122** (1993) 137–152.
13. C. De Felice, Hajós factorizations of cyclic groups - a simpler proof of a characterization, *Journal of Automata, Languages and Combinatorics*, **4** (1999) 111–116.
14. C. De Felice, On some Schützenberger conjectures, submitted to *Inf. and Comp.* (1999), 29 pages.
15. C. De Felice, C. Reutenauer, Solution partielle de la conjecture de factorisation des codes, *Comptes Rendus Acad. Sc. Paris* **302** (1986) 169–170.
16. G. Hajós, Sur la factorisation des groupes abéliens, *Casopis Pest. Mat. Fys.* **74** (1950) 157–162.
17. M. Krasner, B. Ranulac, Sur une propriété des polynômes de la division du cercle, *C. R. Acad. Sc. Paris* **240** (1937) 397–399.
18. N. H. Lam, Hajós factorizations and completion of codes, *Theoret. Comput. Sc.* **182** (1997) 245–256.
19. D. Perrin, Codes asynchrones, *Bull. Soc. math. France* **105** (1977) 385–404.
20. D. Perrin, M. P. Schützenberger, Un problème élémentaire de la théorie de l’information, in: “*Théorie de l’Information*”, *Colloques Internat. CNRS* **276**, Cachan (1977) 249–260.
21. A. Restivo, On codes having no finite completions, *Discrete Math.* **17** (1977) 309–316.
22. C. Reutenauer, Non commutative factorization of variable-length codes, *J. Pure and Applied Algebra* **36** (1985) 167–186.
23. M. P. Schützenberger, Une théorie algébrique du codage, in: *Séminaire Dubreil-Pisot* 1955–56, exposé no 15 (1955) 24 pages.
24. M. Vincent, Construction de codes indécomposables, *RAIRO Inform. Théor.* **19** (1985) 165–178.
25. L. Zhang, C. K. Gu, Two classes of factorizing codes - (p, p) -codes and $(4, 4)$ -codes, in: “*Words, Languages and Combinatorics II*” (M. Ito, H. Jürgensen, eds.), World Scientific (1994) 477–483.

Compositional Characterizations of λ -Terms Using Intersection Types ^{*} (Extended Abstract)

M. Dezani-Ciancaglini¹, F. Honsell², and Y. Motohama¹

¹ Dipartimento di Informatica, Università di Torino
Corso Svizzera 185, 10149 Torino, Italy
{dezani,yoko}@di.unito.it

² Dipartimento di Matematica ed Informatica, Università di Udine
Via delle Scienze 208, 33100 Udine, Italy
honsell@dimi.uniud.it

Abstract. We show how to characterize *compositionally* a number of *evaluation* properties of λ -terms using *Intersection Type* assignment systems. In particular, we focus on termination properties, such as strong normalization, normalization, head normalization, and weak head normalization. We consider also the persistent versions of such notions. By way of example, we consider also another evaluation property, unrelated to termination, namely reducibility to a closed term.

Many of these characterization results are new, to our knowledge, or else they streamline, strengthen, or generalize earlier results in the literature. The completeness parts of the characterizations are proved uniformly for all the properties, using a *set-theoretical* semantics of intersection types over suitable kinds of *stable* sets. This technique generalizes Krivine's and Mitchell's methods for strong normalization to other evaluation properties.

Introduction

The *intersection-types discipline* was introduced in [9] as a tool of overcoming the limitations of Curry's type assignment system. Subsequently it was used in [6] as a tool for proving Scott's conjecture concerning the *completeness* of the set-theoretic semantics for *simple types*.

Very early on, however, it was realized that intersection type theories are a very expressive tool for giving *compositional* characterizations (i.e. characterizations based on properties of proper subterms) of *evaluation* properties of λ -terms. There are two seminal results in this respect.

The first result is that the Ω -free fragment of intersection-types allows one to type *all* and *only* the *strongly normalizing terms*. This is largely a folklore result; the first published proof appears in [21].

^{*} Partially supported by MURST Cofin '99 TOSCA Project and FGV '99.

The second result is the *filter model* construction based on the intersection type theory Σ^{BCD} , carried out in [6]. This result shows that there is a very tight connection between intersection types and *compact elements* in ω -algebraic denotational models of λ -calculus. This connection later received a categorically principled explanation by Abramsky in the broader perspective of “domain theory in logical form”[1].

Since then, the number of intersection type theories, used for the fine study of the denotational semantics of untyped λ -calculus, has increased considerably (e.g. [11,10,16,13,2,20,15]). In all these cases the corresponding intersection type assignment systems are used to provide *finite logical presentations* of particular domain models, which can thereby be viewed also as *filter models*. And hence, intersection type theories provide characterizations of particular semantical properties.

In this paper we address the problem of investigating *uniformly* the use of intersection type theories, and corresponding type assignment systems, for giving a *compositional* characterization of *evaluation* properties of λ -terms.

In particular we discuss *termination* properties such as *strong normalization*, *normalization*, *head normalization*, *weak head normalization*. We consider also the *persistent* versions of such notions (see Definition 8). By way of example we consider also another evaluation property, unrelated to termination, namely reducibility to a *closed term*.

Many of the characterization results that we give are indeed inspired by earlier semantical work on filter models of the untyped λ -calculus, but they are rather novel in spirit. We focus, in fact, on *proof-theoretic* properties of intersection type assignment systems *per se*. Most of our characterizations are therefore new, to our knowledge, or else they streamline, strengthen, or generalize earlier results in the literature.

The *completeness* part of the characterizations is proved uniformly for all the properties. We use a very elementary presentation of the technique of *logical relations* phrased in terms of a *set-theoretical* semantics of intersection types over suitable kinds of *stable* sets. This technique generalizes Krivine’s [17] and Mitchell’s [19] proof methods for *strong normalization*, to other evaluation properties.

The paper is organized as follows. In Section 1 we introduce the intersection type language, intersection type theories and type assignment systems. In Section 2 we introduce the various properties of λ -terms we shall focus on. In Section 3 we give the compositional characterizations of such properties. Final remarks and open problems appear in Section 4.

1 Intersection Type Theories and Type Assignment Systems

Intersection types are syntactical objects which are built inductively by closing a given set $\mathbf{C}$ of *type atoms* (constants) under the *function type* constructor $\rightarrow$ and the *intersection type* constructor $\cap$.

Definition 1 (Intersection type languages). *The intersection type language over $\mathbf{C}$, denoted by $\mathbf{T} = \mathbf{T}(\mathbf{C})$, is defined by the following abstract syntax:*

$$\mathbf{T} = \mathbf{C} \mid \mathbf{T} \rightarrow \mathbf{T} \mid \mathbf{T} \cap \mathbf{T}. \square$$

Upper case Roman letters i.e. $A, B, \dots$, will denote arbitrary types. In writing intersection-types we shall use the following convention: the constructor $\cap$ takes precedence over the constructor $\rightarrow$ and both associate to the right. Moreover $A^n \rightarrow B$ will be short for $\underbrace{A \rightarrow \dots \rightarrow A}_n \rightarrow B$.

Much of the expressive power of intersection type disciplines comes from the fact that types can be endowed with a *preorder relation* $\leq$, which induces the structure of a meet semi-lattice with respect to $\cap$.

Definition 2 (Intersection type preorder). *Let $\mathbf{T} = \mathbf{T}(\mathbf{C})$ be an intersection type language. An intersection type preorder over $\mathbf{T}$ is a binary relation $\leq$ on $\mathbf{T}$ satisfying the following set ∇^0 (“nabla-zero”) of axioms and rules:*

$$\begin{array}{ll} (\text{refl}) & A \leq A \\ (\text{idem}) & A \leq A \cap A \\ (\text{incl}_L) & A \cap B \leq A \\ (\text{incl}_R) & A \cap B \leq B \\ (\text{mon}) & \frac{A \leq A' \quad B \leq B'}{A \cap B \leq A' \cap B'} \quad (\text{trans}) \quad \frac{A \leq B \quad B \leq C}{A \leq C} \square \end{array}$$

We will write $A \sim B$ for $A \leq B$ and $B \leq A$.

Possibly effective, syntactical presentations of intersection type preorders can be given using the notion of *intersection type theory*. An intersection type theory includes always the basic set ∇^0 for $\leq$ and possibly other special purpose axioms and rules.

(Ω)	$A \leq \Omega$
$(\Omega \rightarrow \eta)$	$\Omega \leq \Omega \rightarrow \Omega$
$(\Omega \rightarrow \text{lazy})$	$A \rightarrow B \leq \Omega \rightarrow \Omega$
$(\rightarrow \cap)$	$(A \rightarrow B) \cap (A \rightarrow C) \leq A \rightarrow B \cap C$
(η)	$\frac{A' \leq A \quad B \leq B'}{A \rightarrow B \leq A' \rightarrow B'}$
$(\omega \rightarrow \text{Scott})$	$\Omega \rightarrow \omega \sim \omega$
$(\omega \rightarrow \text{Park})$	$\omega \rightarrow \omega \sim \omega$
$(\omega \varphi)$	$\omega \leq \varphi$
$(\varphi \rightarrow \omega)$	$\varphi \rightarrow \omega \sim \omega$
$(\omega \rightarrow \varphi)$	$\omega \rightarrow \varphi \sim \varphi$

Fig. 1. Some special purpose Axioms and Rules concerning $\leq$.

Definition 3 (Intersection type theories). Let $\mathbf{T} = \mathbf{T}(\mathbf{C})$ be an intersection type language, and let ∇ be a set of axioms and rules for deriving judgments of the shape $A \leq B$, with $A, B \in \mathbf{T}$. The intersection type theory $\Sigma(\mathbf{C}, \nabla)$ is the set of all judgments $A \leq B$ derivable from the axioms and rules in $\nabla^0 \cup \nabla$. $\square$

When we consider the intersection type theory $\Sigma(\mathbf{C}, \nabla)$, we will write:

$$\mathbf{C}^\nabla \text{ for } \mathbf{C}, \quad \mathbf{T}^\nabla \text{ for } \mathbf{T}(\mathbf{C}), \quad \Sigma^\nabla \text{ for } \Sigma(\mathbf{C}, \nabla).$$

Moreover $A \leq_\nabla B$ will be short for $(A \leq B) \in \Sigma^\nabla$. Finally we will write $A \sim_\nabla B$ for $A \leq_\nabla B \leq_\nabla A$.

In Figure 1 appears a list of special purpose axioms and rules which have been considered in the literature. We refer to [7,12] for motivations.

The element Ω plays a very special role in the development of the theory. Therefore we stipulate the following blanket assumption:

$$\text{if } \Omega \in \mathbf{C}^\nabla \text{ then } (\Omega) \in \nabla.$$

$\mathcal{C}^a$	$= \mathbf{C}_\infty$	$\mathcal{B}a$	$= \{(\rightarrow, \cap), (\eta)\}$
$\mathcal{C}^{\mathcal{AO}}$	$= \{\Omega\}$	$\mathcal{AO}$	$= \mathcal{B}a \cup \{(\Omega), (\Omega\text{-lazy})\}$
$\mathcal{C}^{\mathcal{BCD}}$	$= \{\Omega\} \cup \mathbf{C}_\infty$	$\mathcal{BCD}$	$= \mathcal{B}a \cup \{(\Omega), (\Omega\text{-}\eta)\}$
$\mathcal{C}^{\mathcal{S}^c}$	$= \{\Omega, \omega\}$	$\mathcal{S}^c$	$= \mathcal{BCD} \cup \{(\omega\text{-Scott})\}$
$\mathcal{C}^{\mathcal{P}^a}$	$= \{\Omega, \omega\}$	$\mathcal{P}^a$	$= \mathcal{BCD} \cup \{(\omega\text{-Park})\}$
$\mathcal{C}^{\mathcal{CDZ}}$	$= \{\Omega, \varphi, \omega\}$	$\mathcal{CDZ}$	$= \mathcal{BCD} \cup \{(\omega\varphi), (\varphi \rightarrow \omega), (\omega \rightarrow \varphi)\}$

Fig. 2. Type Theories: atoms, axioms and rules.

We introduce in Figure 2 a list of significant intersection type theories which will be handy in the characterizations given in Section 3. The order is logical, rather than historical: [5,2,6,22,16,10]. We shall denote such theories as Σ^∇ with various different names ∇ , picked for mnemonic reasons. For each such ∇ we specify in Figure 2 the type theory $\Sigma^\nabla = \Sigma(\mathbf{C}, \nabla)$ by giving the set of constants $\mathbf{C}^\nabla$ and the set ∇ of extra axioms and rules taken from Figure 1. Here $\mathbf{C}_\infty$ is an infinite set of fresh atoms, i.e. different from Ω, φ, ω .

Before giving the crucial notion of *intersection type assignment system*, we need to introduce some preliminary definitions and notations.

Definition 4. i) A ∇ -basis is a set of statements of the shape $x:B$, where $B \in \mathbf{T}^\nabla$, all whose variables are distinct.
 ii) An intersection type assignment system $\lambda \cap^\nabla$ relative to Σ^∇ is a formal system for deriving judgments of the form $\Gamma \vdash^\nabla M : A$, where the subject M is an untyped λ -term, the predicate A is in $\mathbf{T}^\nabla$, and Γ is a ∇ -basis. $\square$

Definition 5 (Basic Type Assignment System). Let Σ^∇ be a type theory. The basic type assignment system $\lambda\cap_{\mathbf{B}}^\nabla$ is a formal system for deriving judgments of the shape $\Gamma \vdash_{\mathbf{B}}^\nabla M : A$. Its rules are the following

$$\begin{array}{c}
(Ax) \frac{x:A \in \Gamma}{\Gamma \vdash_{\mathbf{B}}^\nabla x:A} \qquad (\rightarrow I) \frac{\Gamma, x:A \vdash_{\mathbf{B}}^\nabla M : B}{\Gamma \vdash_{\mathbf{B}}^\nabla \lambda x.M : A \rightarrow B} \\
(\rightarrow E) \frac{\Gamma \vdash_{\mathbf{B}}^\nabla M : A \rightarrow B \quad \Gamma \vdash_{\mathbf{B}}^\nabla N : A}{\Gamma \vdash_{\mathbf{B}}^\nabla MN : B} \qquad (\cap I) \frac{\Gamma \vdash_{\mathbf{B}}^\nabla M : A \quad \Gamma \vdash_{\mathbf{B}}^\nabla M : B}{\Gamma \vdash_{\mathbf{B}}^\nabla M : A \cap B} \\
(\leq_\nabla) \frac{\Gamma \vdash_{\mathbf{B}}^\nabla M : A \quad A \leq_\nabla B}{\Gamma \vdash_{\mathbf{B}}^\nabla M : B} \square
\end{array}$$

If $\Omega \in \mathbf{C}^\nabla$, in line with the intended set-theoretic interpretation of Ω as the universe, we extend the Basic Type Assignment System with a suitable axiom for Ω :

Definition 6 (Ω -type Assignment System). Let Σ^∇ be a type theory with $\Omega \in \mathbf{C}^\nabla$. The axioms and rules of the Ω -type assignment system $\lambda\cap_{\Omega}^\nabla$ are those of the Basic type Assignment System, together with the further axiom:

$$(Ax-\Omega) \quad \Gamma \vdash_{\Omega}^\nabla M : \Omega. \square$$

For ease of notation, we assume that the symbol Ω is reserved for the type constant used in the system $\lambda\cap_{\Omega}^\nabla$, and hence we forbid $\Omega \in \mathbf{C}^\nabla$ when we deal with $\lambda\cap_{\mathbf{B}}^\nabla$. In the following $\lambda\cap^\nabla$ will range over $\lambda\cap_{\mathbf{B}}^\nabla$ and $\lambda\cap_{\Omega}^\nabla$. More precisely we convene that $\lambda\cap^\nabla$ stands for $\lambda\cap_{\Omega}^\nabla$ whenever $\Omega \in \mathbf{C}^\nabla$, and for $\lambda\cap_{\mathbf{B}}^\nabla$ otherwise. Similarly for $\vdash^\nabla$.

2 Some Distinguished Properties of λ -Terms

In this section we introduce the distinguished classes of λ -terms which we shall focus on in this paper.

We shall consider first *termination properties*. In particular we shall discuss the crucial property of being *strongly normalizing* and the three properties of having a β -normal form, of having a head normal form, and of having a weak head normal form.

Definition 7 (Normalization property).

- i) M is a normal form, $M \in \mathbf{NF}$, if M cannot be further reduced;
- ii) M is strongly normalizing, $M \in \mathbf{SN}$, if all reductions starting at M are finite;
- iii) M has a normal form, $M \in \mathbf{N}$, if M reduces to a normal form;
- iv) M has a head normal form, $M \in \mathbf{HN}$, if M reduces to a term of the form $\lambda x.y\mathbf{M}$ (where possibly y appears in x);
- v) M has a weak head normal form, $M \in \mathbf{WN}$, if M reduces to an abstraction or to a term starting with a free variable. $\square$

For each of the above properties, but **SN**, in the above definition, we shall consider also the corresponding *persistent* version (see Definition 8). *Persistently normalizing* terms have been introduced in [8].

Definition 8 (Persistent normalization property).

- i) A term M is persistently normalizing, $M \in \text{PN}$, if $MN \in \text{N}$ for all terms N in N .
- ii) A term M is a persistently normalizing normal form, $M \in \text{PNF}$, if $M \in \text{PN} \cap \text{NF}$.
- iii) A term M is persistently head normalizing, $M \in \text{PHN}$, if $MN \in \text{HN}$ for all terms N .
- iv) A term M is persistently weak normalizing, $M \in \text{PWN}$, if $MN \in \text{WN}$ for all terms N . $\square$

Figure 3 illustrates mutual implications between the above notions.

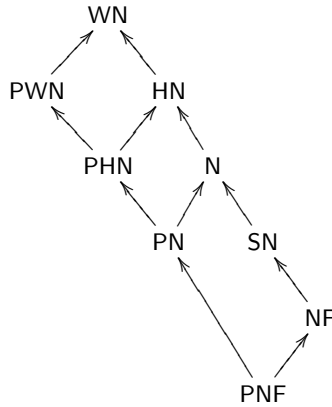


Fig. 3. Inclusions between sets of λ -terms.

Intersection types can be used to characterize *compositionally* also other evaluation properties of terms, which are not linked to termination. In this paper we shall consider, by way of example, the property of *reducing to a closed term*. Hence we conclude this section with the definition of:

Definition 9 (Closable term). M is closable, $M \in \text{C}$, if M reduces to a closed term. $\square$

3 Characterizing Compositionally Properties of λ -Terms

In this section we put to use *intersection type disciplines* to give a *compositional* characterization of evaluation properties of λ -terms.

In this section we give the main result of the paper, Theorem 1. For each of the properties introduced in Section 2, Theorem 1 provides *compositional* characterizations in terms of intersection type assignment systems.

Some of the properties characterized in Theorem 1 had received already a characterization in terms of intersection type disciplines. The most significant case is that of strongly normalizing terms. One of the original motivations for introducing intersection types in [21] was precisely that of achieving such a characterization. Alternative characterizations appear in [18,5,17,14,4,15]. In [10] both normalizing and persistently normalizing terms had been characterized using intersection types. Closed terms were characterized in [16]. The characterizations appearing in Theorem 1 strengthen and generalize all earlier results, since all mentioned papers consider only specific type theories, and hence in our view Theorem 1 appears more intrinsic.

Before giving the main theorem a last definition is necessary.

Definition 10. A type theory Σ^∇ is an arrow-type theory if $\Omega \in \mathbf{C}^\nabla$ and for all $\psi \in \mathbf{C}^\nabla$ either $\psi \sim_{\nabla} \Omega$ or $\exists I, \{A_i, B_i\}_{i \in I}. \psi \sim_{\nabla} \bigcap_{i \in I} (A_i \rightarrow B_i)$. $\square$

Of the various type theories appearing in Figure 2 only Σ^{Sc} , $\Sigma^{\mathcal{P}a}$, and Σ^{CDZ} are arrow-type theories.

Finally we can state the main result:

Theorem 1 (Characterization).

1 Normalization properties

- i) (strongly normalizing terms) A λ -term $M \in \mathbf{SN}$ if and only if for all type theories Σ^∇ there exist $A \in \mathbf{T}^\nabla$ and a ∇ -basis Γ such that $\Gamma \vdash^\nabla M : A$. Moreover in the system $\lambda \cap_{\Omega}^{\mathcal{B}a}$ the terms satisfying the latter property are precisely the strongly normalizing ones.
- ii) (normalizing terms) A λ -term $M \in \mathbf{N}$ if and only if for all type theories Σ^∇ such that $\{\Omega\} \subset \mathbf{C}^\nabla$, there exist $A \in \mathbf{T}^\nabla$ and a ∇ -basis Γ such that $\Gamma \vdash_{\Omega}^\nabla M : A$ and $\Omega \notin A, \Gamma$. Moreover in the system $\lambda \cap_{\Omega}^{\mathcal{B}CD}$ the terms satisfying the latter property are precisely the ones which have a normal form. Furthermore, in the system $\lambda \cap_{\Omega}^{\mathcal{C}DZ}$ the terms typable with type φ , in the $\mathcal{C}DZ$ -basis all of whose predicates are ω , are precisely the ones which have a normal form.
- iii) (head normalizing terms) A λ -term $M \in \mathbf{HN}$ if and only if for all type theories Σ^∇ such that $\Omega \in \mathbf{C}^\nabla$, and for all $A \in \mathbf{T}^\nabla$ there exist a ∇ -basis Γ and two integers m, n such that $\Gamma \vdash_{\Omega}^\nabla M : (\Omega^m \rightarrow A)^n \rightarrow A$. Moreover in the system $\lambda \cap_{\Omega}^{\mathcal{B}CD}$ the terms satisfying the latter property are precisely the ones which have a head normal form.
- iv) (weak head normalizing terms) A λ -term $M \in \mathbf{WN}$ if and only if for all type theories Σ^∇ such that $\Omega \in \mathbf{C}^\nabla$, there exists a ∇ -basis Γ such that $\Gamma \vdash_{\Omega}^\nabla M : \Omega \rightarrow \Omega$. Moreover in the system $\lambda \cap_{\Omega}^{\mathcal{A}O}$ the terms satisfying the latter property are precisely the ones which have a weak head normal form.

2 Persistent normalization properties

- i) (*persistently normalizing terms*) A λ -term $M \in \text{PN}$ if and only if for all arrow-type theories Σ^∇ and all $A \in \mathbf{T}^\nabla$ there exists a ∇ -basis Γ such that $\Gamma \vdash_\Omega^\nabla M : A$. Moreover in the system $\lambda\cap_\Omega^{\text{CDZ}}$ the terms typable with type ω , in the CDZ -basis all of whose predicates are ω , are precisely the persistently normalizing ones.
- ii) (*persistently head normalizing terms*) A λ -term $M \in \text{PHN}$ if and only if for all type theories Σ^∇ such that $\Omega \in \mathbf{C}^\nabla$ and all $A \in \mathbf{T}^\nabla$ there exist a ∇ -basis Γ and an integer n such that $\Gamma \vdash_\Omega^\nabla M : \Omega^n \rightarrow A$. Moreover in the system $\lambda\cap_\Omega^{\text{Sc}}$ the terms typable with type ω , in the Sc -basis all of whose predicates are ω , are precisely the persistently head normalizing ones.
- iii) (*persistently weak normalizing terms*) A λ -term $M \in \text{PWN}$ if and only if for all type theories Σ^∇ such that $\Omega \in \mathbf{C}^\nabla$ and all integers n there exists a ∇ -basis Γ such that $\Gamma \vdash_\Omega^\nabla M : \Omega^n \rightarrow \Omega$. Moreover in the system $\lambda\cap_\Omega^{\text{AO}}$ the terms satisfying the latter property are precisely the persistently weak normalizing ones.

3 Closability (*closed terms*) A λ -term $M \in \mathbf{C}$ if and only if for all type theories Σ^∇ such that $\Omega \in \mathbf{C}^\nabla$ and $\omega \sim_\nabla \omega \rightarrow \omega$ for some $\omega \in \mathbf{C}^\nabla$, M is typable with type ω in the empty ∇ -basis. Moreover in the system $\lambda\cap_\Omega^{\text{Pa}}$ the terms satisfying the latter property are precisely the terms which reduce to closed terms. $\square$

The proofs of the only if parts of the Theorem are mainly straightforward inductions and case splits, but for the case of persistently normalizing terms. The proofs of the if parts require the set-theoretic semantics of intersection types using stable sets [3,7].

The set-theoretic semantics of a type, given an applicative structure, is a subset of the structure itself. *Intersection* is interpreted as *set-theoretic intersection*, $\leq$ is interpreted as *set-theoretic inclusion*, and $A \rightarrow B$ is interpreted *à la logical relation*, i.e. as a subset of the points of the structure whose functional behaviour is that of mapping all points in A into B .

In the present context, there is only one applicative structure under consideration. This is the *term structure* Λ , i.e. the applicative structure whose domain are the λ -terms and where application is just juxtaposition of terms.

In order to ensure that the interpretations of types consist of terms which satisfy appropriate properties, we need to give the set-theoretic semantics using special classes of *stable* sets, for suitable notions of stability. These stability properties amount essentially to suitable invariants for the set-theoretic operators corresponding to the type constructors.

For lack of space we omit these proofs which can be found in [12].

4 Concluding Remarks

Two natural questions, at least, lurk behind this paper: “can we characterize in some significant way the class of evaluation properties which we can characte-

size using intersection types?” and “is there a method for going from a logical specification of a property to the appropriate intersection type theory?”.

Regarding the first question, we have seen that the properties have to be closed, at least, under some form of β -expansion. But clearly this is not the whole story. Probably the answer to this question is linked to some very important open problems in the theory of the denotational semantics of untyped λ -calculus, like the existence of a denotational model whose theory is precisely $\lambda\beta$. As far as the latter question is concerned, we really have no idea. It seems that we are still missing something in our understanding of intersection types.

Of course there are some partial answers. For instance by looking at what happens in particular filter models, one can draw some inspiration and sometimes even provide some interesting characterizations. In this paper we discussed closable sets. Another example would have been, for instance, that of those terms which *reduce to terms of the λ -I-calculus*. Here the filter model under consideration is the one in [16], generated by the theory $\Sigma^{\mathcal{HR}} = (\{\Omega, \varphi, \omega\}, \mathcal{BCD} \cup \{(\omega\varphi), (\varphi \rightarrow \omega), (\omega \rightarrow I)\})$, where $(\omega \rightarrow I)$ is the rule $(\varphi \rightarrow \varphi) \cap (\omega \rightarrow \omega) \sim \varphi$. The terms typable with φ in $\lambda\cap_{\mathcal{B}}^{\mathcal{HR}}$, in an environment where all variables have type φ , are then precisely those which reduce to terms of the λ -I-calculus [16]. These characterizations however appear quite *accidental*. And we feel that we lack yet a general theory which could allow us to streamline the approach. Given the model we can start to guess. And when we are successful, as in this case, we can achieve generality only artificially, by considering all those type theories which extend the theory of the filter model in question.

For one thing this method of drawing inspiration from filter models is interesting, in that it provides some very interesting conjectures. Perhaps the best example concerns *persistently strongly normalizing* terms. These are those strongly normalizing terms M , such that for all vectors $\mathbf{N}$ of strongly normalizing terms, $M\mathbf{N}$ is still strongly normalizing. Consider the filter model introduced in [15], generated by the type theory obtained by pruning the type theory $\Sigma^{\mathcal{CDZ}}$ of all types including Ω , i.e. generated by the theory $\Sigma^{\mathcal{HL}} = (\{\varphi, \omega\}, \mathcal{Ba} \cup \{(\omega\varphi), (\varphi \rightarrow \omega), (\omega \rightarrow \varphi)\})$. The natural conjecture is then, in analogy to what happens for persistently normalizing terms, “the terms typable with ω in $\lambda\cap_{\mathcal{B}}^{\mathcal{HL}}$, in the $\mathcal{HL}$ -basis where all variables have type ω , are precisely the *persistently strongly normalizing* ones”. Completeness is clear, but to show soundness some independent syntactical characterization of that class of terms appears necessary. The set of persistently strongly normalizing terms does not include $\text{PN} \cap \text{SN}$. A counter example is $M \equiv \lambda x.a((\lambda y.b)(xx))$ since $M(\lambda z.zz) \notin \text{SN}$. This conjecture still resists proof.

References

1. S. Abramsky. Domain theory in logical form. *Ann. Pure Appl. Logic*, 51(1-2):1–77, 1991.
2. S. Abramsky and C.-H. L. Ong. Full abstraction in the lazy lambda calculus. *Inform. and Comput.*, 105(2):159–267, 1993.

3. F. Alessi, M. Dezani-Ciancaglini, and F. Honsell. A complete characterization of the complete intersection-type theories. In *Proceedings in Informatics*. ITRS'00 Workshop, Carleton-Scientific, 2000.
4. R. M. Amadio and P.-L. Curien. *Domains and lambda-calculi*. Cambridge University Press, Cambridge, 1998.
5. S. van Bakel. Complete restrictions of the intersection type discipline. *Theoret. Comput. Sci.*, 102(1):135–163, 1992.
6. H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J. Symbolic Logic*, 48(4):931–940, 1983.
7. H.P. Barendregt et. al. *Typed λ -calculus and applications*. North-Holland. (to appear).
8. C. Böhm and M. Dezani-Ciancaglini. λ -terms as total or partial functions on normal forms. In C. Böhm, editor, *λ -calculus and computer science theory*, pages 96–121, Lecture Notes in Comput. Sci., Vol. 37. Springer, Berlin, 1975.
9. M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ -calculus. *Notre Dame J. Formal Logic*, 21(4):685–693, 1980.
10. M. Coppo, M. Dezani-Ciancaglini, and M. Zacchi. Type theories, normal forms, and D_∞ -lambda-models. *Inform. and Comput.*, 72(2):85–116, 1987.
11. M. Coppo, F. Honsell, M. Dezani-Ciancaglini, and G. Longo. Extended type structures and filter lambda models. In G. Longo et. al., editors, *Logic colloquium '82*, pages 241–262. North-Holland, Amsterdam, 1984.
12. M. Dezani-Ciancaglini, F. Honsell, and Y. Motohama. Compositional characterization of λ -terms using intersection types. Internal report, 2000, (<http://www.di.unito.it/~yoko/paper/dezahonsmoto2000.ps>).
13. L. Egidi, F. Honsell, and S. Ronchi Della Rocca. Operational, denotational and logical descriptions: a case study. *Fund. Inform.*, 16(2):149–169, 1992.
14. S. Ghilezan. Strong normalization and typability with intersection types. *Notre Dame J. Formal Logic*, 37(1):44–52, 1996.
15. F. Honsell and M. Lenisa. Semantical analysis of perpetual strategies in λ -calculus. *Theoret. Comput. Sci.*, 212(1-2):183–209, 1999.
16. F. Honsell and S. Ronchi Della Rocca. An approximation theorem for topological lambda models and the topological incompleteness of lambda calculus. *J. Comput. System Sci.*, 45(1):49–75, 1992.
17. J.-L. Krivine. *Lambda-calcul Types et Modèles*. Masson, Paris, 1990. English translation: *Lambda-calculus, types and models*, Ellis Horwood, 1993.
18. D. Leivant. Typing and computational properties of lambda expressions. *Theoret. Comput. Sci.*, 44(1):51–68, 1986.
19. J. Mitchell. *Foundation for Programming Languages*. MIT Press, 1996.
20. G. D. Plotkin. Set-theoretical and other elementary models of the λ -calculus. *Theoret. Comput. Sci.*, 121(1-2):351–409, 1993.
21. G. Pottinger. A type assignment for the strongly normalizable λ -terms. In J.R. Hindley and J.P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–577. Academic Press, London, 1980.
22. D. Scott. Continuous lattices. In F.W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, pages 97–136. Lecture Notes in Math., Vol. 274. Springer, Berlin, 1972.

Time and Message Optimal Leader Election in Asynchronous Oriented Complete Networks

Stefan Dobrev*

Institute of Mathematics, Slovak Academy of Sciences
Department of Informatics
P.O. Box 56, 840 00 Bratislava, Slovak Republic
E-mail: stefan@ifi.savba.sk

Abstract. We consider the problem of leader election in asynchronous oriented N -node complete networks. We present a leader election algorithm with $O(N)$ message and $O(\log \log N)$ time complexity. The message complexity is optimal and the time complexity is the best possible under the assumption of message optimality.

The best previous leader election algorithm for asynchronous oriented complete networks by Singh [16] achieves $O(N)$ message and $O(\log N)$ time complexity.

1 Introduction

Leader election is a fundamental problem in distributed computing and has a number of applications. It is an important tool for breaking symmetry in a distributed system. By distinguishing a single node as a leader, it is possible to execute centralized protocols in decentralized environments. Many problems such as spanning-tree construction, extrema finding or computing a global function are equivalent to leader election in terms of their message and time complexities, hence they benefit from improvements in election algorithms as well.

The problem of leader election has been extensively studied on wide variety of models and topologies, including arbitrary networks, rings, chordal rings, tori, hypercubes and other topologies, with a broad range of available structural information and different models of communication. Most of the attention has been given to the study of message complexity. However, as message-optimal solutions have been achieved, a natural question arises to improve the time complexity as well, while keeping the message complexity optimal.

The best upper and lower bounds for several widely used topologies are shown in the Table 1. The distinction between unoriented and oriented case for hypercubes and complete networks is based on the amount of structural knowledge available to processors. In the unoriented case the processors do not know where their incident links lead to. In oriented hypercubes, a processor knows for each incident link the dimension it lies in. Oriented complete graphs

* Supported by VEGA grant No. 2/7007/20

mean a Hamiltonian circle is chosen and each processor knows how far ahead in this circle each of its incident links leads to – see Figure 1.

The upper bounds on time are for message-optimal algorithms. In most cases the network diameter is the lower bound for the time complexity, and this bound is often achievable, even by message optimal algorithms. Complete graphs are an exception from this rule. The fastest algorithm (each processor broadcasts its identity and then chooses the smallest one as the leader) is of $O(N^2)$ message complexity, far from optimal. Singh in [15] proved that any election algorithm for asynchronous unoriented complete networks of message complexity Nk is of time complexity at least $N/16k$. Combined with the lower bound of $\Omega(N \log N)$ on message complexity for election algorithms in unoriented complete graphs, this implies $\Omega(N/\log N)$ lower bound on the time complexity for message optimal algorithms. Better results, both in message and time complexities, are achievable in the oriented case. Loui, Matsushita and West in [12] give an $O(N)$ message election algorithm of $O(N)$ time complexity. Singh in [16] improved their result, showing an $O(N)$ message algorithm of time complexity $O(\log N)$. Israeli, Kranakis, Krizanc and Santoro in [10] prove an $\Omega(N^{2^t/(2^t-1)})$ lower bound on message complexity for any t -step wake-up algorithm for oriented complete graphs. This implies an $\Omega(\log \log N)$ lower bound on time for any $O(N)$ message election protocol in oriented complete graphs.

Table 1. Previous results

Network topology	Lower bound – message	Upper bound – message	Upper bound – time
Arbitrary	$\Omega(N \log N + E)$ [8]	$O(N \log N + E)$ [8]	$O(N)$ [1]
Rings	$\Omega(N \log N)$ [2]	$O(N \log N)$ [9]	$O(N)$
Complete			
– unoriented	$\Omega(N \log N)$ [11]	$O(N \log N)$ [11]	$O(N/\log N)$ [15]
– oriented	$\Omega(N)$	$O(N)$ [12]	$O(\log N)$ [16]
Hypercubes			
– unoriented	$\Omega(N)?$	$O(N \log \log N)$ [3]	$O(N)$ [3]
– oriented	$\Omega(N)$	$O(N)$ [6,19]	$O(\log^2 N)$ [19] ¹
Tori	$\Omega(N)$	$O(N)$ [13,14]	$O(\sqrt{N})$ [13,14]
Butterflies, CCC	$\Omega(N)$	$O(N)$ [4]	$O(\log^2 N)$ [4]

In this paper, we fill the gap between the $O(\log N)$ upper bound from [16] and the $\Omega(\log \log N)$ lower bound from [10] by providing a leader election algorithm for asynchronous oriented complete graphs, which is of optimal $O(N)$ message complexity and simultaneously achieves optimal $O(\log \log N)$ time complexity as well.

This paper is organized as follows. In section 2 we introduce the model. In section 3 we present the algorithm and analyze its complexities. Finally, concluding remarks can be found in section 4.

¹ A recent result in [5] improves this to $O(\log N)$.

2 Model

We consider the standard model of asynchronous distributed computing in point-to-point networks [18]. Communication is achieved by message passing. Asynchronous communication means that every message is delivered in a finite but unbounded time, while the time for local computation is considered negligible. We assume a processor can simultaneously send/receive messages to/from all its neighbours (so called *shouting* or *all-port* model). FIFO requirements on links are not necessary – messages sent via the same link can take-over each other. All processors are identical and run the same algorithm.

We consider the communication topology to be the N -node complete network. The processors are denoted by $p_0, p_1, \dots, p_{N-1}$, numbered counter-clockwise in the ring (chosen Hamiltonian circle of the network). We will use *ring distance* to mean the distance of two processors in this chosen ring. The network is oriented in the sense that chordal sense of direction [7] is given: The link leading from p_i to p_j is labeled by $j - i \bmod N$ at p_i and by $i - j \bmod N$ at p_j – see Figure 1.

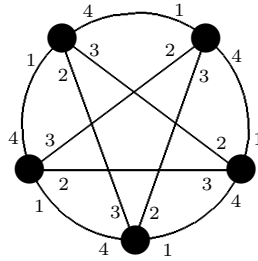


Fig. 1. Oriented complete graph

We consider non-anonymous networks. That means that each processor has an unique identification number (*id*) from some totally ordered set ID . These identities are without topological significance. Moreover, each processor knows only its own identity.

To solve the problem of *leader election* means that starting from an initial configuration with all processors in the same state (the only difference being different *ids*), the system should reach a configuration where exactly one processor is in the state *leader* and all other processors are in the state *defeated*. The computation starts spontaneously at some non-empty subset of processors. The remaining processors join the computation either after receiving the first message or by spontaneous wake-up sometimes later.

The *message complexity* of an algorithm is the maximum number of messages sent during any possible execution of the algorithm. Note that there are many possible executions due to different wake-up schedules and link delays. We assume messages to be of size $O(\log N)$ bits and assume all *ids* can be coded within that space (so a unit message can carry an identity of a processor).

The *time complexity* is the worst-case execution time, assuming delivery of a message takes at most one time unit. All local computation is supposed to take negligible time.

3 The Algorithm *Target&Bullets*

In this section we present a leader election algorithm *Target&Bullets* for asynchronous oriented complete graphs. In the analysis we show it is of optimal $O(N)$ message and $O(\log \log N)$ time complexity.

The execution of *Target&Bullets* proceeds in logical rounds. Each spontaneously awoken processor – leader candidate – starts in round 1. In a round k a candidate v tries to claim a group of nearby processors as its *domain*. If a stronger candidate is encountered during this process, v is defeated and participates only passively in the rest of the computation. Otherwise v is promoted to round $k + 1$ and proceeds by trying to claim a larger domain. The computation terminates when v successfully claims a domain of size more than $N/2$ – there is no possibility other candidate can claim such a big domain.

The main idea for keeping the message complexity of *Target&Bullets* linear is to be able to claim a domain, while spending much less than the size of the domain messages. This is not a new approach, it has been used in election algorithms since mid-80's, see [4,14,19]. The optimal time complexity of *Target&Bullets* is due to rapid growth of the size of the domains.

Let S_k be a parameter related to the size of the domains of candidates in round k . The goal of a candidate v in round k is to claim a domain of size S_k^2 , while spending $O(S_k)$ messages (we will show how to do it later). Claiming a domain means ensuring that no other candidate in round k claims a processor from that domain. It follows at most N/S_k^2 candidates can successfully claim domains in round k and proceed to round $k + 1$. Since a candidate in round $k + 1$ ($k \geq 1$) spends $O(S_{k+1})$ messages, if we choose $S_{k+1} = S_k^2/2^k$, the cost of the round $k + 1$ can be bounded by $O(S_{k+1}N/S_k^2) = O(N/2^k)$. Choosing $S_1 \in O(1)$ and summing for all rounds (taking into account that the cost of the first round is $O(S_1N) \in O(N)$) results in $O(N)$ overall message complexity. Note that

$$S_k = \frac{S_1^{2^{k-1}}}{2^{\sum_{i=1}^{k-1} 2^{k-1-i}}} = \frac{S_1^{2^{k-1}}}{2^{2^k - k - 1}}$$

Choosing $S_1 = 8$ yields $S_k = 2^{2^{k-1}2^{k+1}}$, with $S_k \in O(2^{2^k})$ and $S_k \in \Omega(2^{2^{k-1}})$. That means $\log \log N + 1$ rounds are enough to elect the leader.

A domain of size S_k^2 can be claimed by first marking a block of the closest S_k processors in the ring (marking a *target* set) and then marking S_k processors to the left and right with stride S_k (firing *bullets*) – see Figure 2. Since the bullets are fired with stride S_k , no target set of another candidate in round k (its size is S_k) whose ring distance is at most S_k^2 can slip between the bullets. Hence if two candidates in the round k do not collide, their ring distance is more than S_k^2 . If two candidates collide, at most one of them survives – the one which collided

only with weaker candidates. (Note that a candidate can simultaneously collide with several candidates.) A candidate is weaker if it is either in a smaller round, or it is in the same round, but its identity is lower.

Due to asynchronous communication careful processing is necessary to ensure proper claiming of domains. It may happen that a bullet of a candidate v with id_v smaller than id_u did not find a target of a candidate u , although their ring distance is at most S_k^2 . This may be the case if the target messages of u have not yet been delivered, although u and v are in the same round. In such a scenario both u and v would be promoted to the next round – v because it was faster and has not seen u , u because it has seen only weaker candidates. The problem is solved using the two-phase technique from [5]. The idea is to first mark the target set and only after it went successfully, fire the bullets. Strength of candidates is modified to take into account the phase they are in – a candidate firing bullets is considered stronger than a candidate marking its target set (if they are in the same round), even if the latter has higher identity. Moreover, all the processors in the target set are acknowledged whether the candidate survived the marking. In this way, when a bullet message comes to a processor marked by a *target* message, it learns (possibly waiting for the acknowledgement) whether a real collision with alive candidate occurs. In the above scenario u would not survive marking the target set – it would find a *bullet* message of v , which is now stronger than its *target* messages.

Algorithm *Target&Bullets*

Messages used: ("type", id , r), where "type" is one of *target*, *bullet*, *die!*, *OK*, *alive*, and *dead*. id is from the set ID of processor identities and r is the round number.

Local variables at a processor v : Each processor remembers the last message of types *target*, *alive* and *dead*. In addition it remembers (in variable M) the strongest message it has seen. (When two messages are compared, first the round fields are considered, then the type (*bullet* being stronger than *target*), then the id fields.) To make the code more readable, updating these variables is omitted from it.

The algorithm for a spontaneously awoken processor v , starting with $r = 0$:

do

$r := r + 1$; { *The round number.* }

Send (*target*, id , r) via links $\pm 1, \pm 2, \dots, \pm S_r/2$; { *Mark the target set.* }

Wait for *die!* and/or *OK* messages from all links $\pm 1, \pm 2, \dots, \pm S_r/2$;

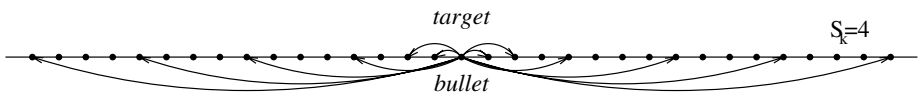


Fig. 2.

if a *die!* message has been received then

Send (*dead*, *id*, *r*) via links $\pm 1, \pm 2, \dots, \pm S_r/2$; {*Acknowledge death.*}
terminate.

else

Send (*alive*, *id*, *r*) via links $\pm 1, \pm 2, \dots, \pm S_r/2$; {*Acknowledge survival.*}
 {*Fire the bullets.*}

Send (*bullet*, *id*, *r*) via links $\pm S_r, \pm 2S_r, \dots, \pm S_r^2$;

Wait for *die!* and/or *OK* messages from all links $\pm S_r, \pm 2S_r, \dots, \pm S_r^2$;

if a *die!* message has been received then

terminate.

fi;

fi;

{*Survived round r.*}

while $S_r^2 \leq N/2$;

Send (*Leader: id*) to all processors; {*v is the leader, broadcast its identity.*}

It may happen a processor simultaneously receives several messages. In that case, the incoming messages are processed one-by-one, in an arbitrary order. In addition, the candidate processor reacts to the *bullet* messages as if it has received it's own *target* message.

At a processor *v* after receiving message $m = (\textit{target}, \textit{id}, r)$ via link *l*:

if *m* is weaker than *M* then {*Weaker than the strongest message seen by v.*}

Send (*die!*) via *l*;

else

Send (*OK*) via *l*;

fi;

At a processor *v* after receiving message $m = (\textit{bullet}, \textit{id}, r)$ via link *l*:

if *m* is weaker than *M* then {*Weaker than the strongest message seen by v.*}

Send (*die!*) via *l*;

else

if *v* has received a message (*target*, *id'*, *r*) with $\textit{id}' < \textit{id}$ then

Wait until either (*dead*, *id'*, *r*) or (*alive*, *id'*, *r*) is received;

if (*alive*, *id'*, *r*) was received then

Send (*die!*) via *l*;

else

Send(*OK*) via *l*;

fi;

else

Send (*OK*) via *l*;

fi;

fi;

Note that since $S_{k+1} < S_k^2$, except for the first round, no processor receives *target* messages of the same round from different candidates. That means that with exception of the first round (where $S_1 = 8$ of *target/alive/dead* messages are to be remembered), it is enough to remember only the last *target/alive/dead* message to properly implement handling of the *bullet* messages.

Lemma 1. *Let u and v be two candidates which survived round r . Then their ring distance is more than S_r^2 .*

Proof. Let the ring distance of u and v be at most S_r^2 . Then there is a processor w which received a *target* message from u and a *bullet* message from v . Since u survived, w has received the *target* message from u before the *bullet* message from v . Consider now the handling of the *bullet* message at w . If the identity of v is lower than the identity of u , a *die!* message is replied to v and v does not survive the round r . Hence the identity of v is higher than the identity of u . This means the replying to its *bullet* message is delayed until a *dead* or *alive* message arrives for u . Since u survived round r , an *alive* message arrived, forcing a *die!* reply to v . Contradiction with the assumption that both u and v survived round r .

Theorem 1. *The algorithm Target&Bullets elects a leader in asynchronous oriented complete networks in time $O(\log \log N)$, using $O(N)$ messages.*

Proof. Correctness. Let a processor v declares itself as the leader in round r . Since $S_r^2 > N/2$, it follows from Lemma 1 that it is the single candidate surviving round r . We should prove that always at least one candidate remains. Let $r' \leq r$ be the smallest round such that no candidate survived round r' . Suppose no candidate survived marking the target set. Consider the candidate v with the highest identity among the candidates surviving round $r' - 1$. Since no message with round higher than r' was sent, neither any *bullet* message for round r' , the *target* messages of v are the strongest messages sent and v will survive marking its target set. Hence there must be some candidates surviving marking their target sets. Consider now the candidate v with the highest identity among these candidates. Since no candidate reaches round $r' + 1$, the *bullet* messages of v are the strongest messages. Hence v receives only *OK* replies and is promoted to round $r' + 1$. Contradiction.

Communication complexity. A candidate in round r sends S_r *target* messages and possibly $2S_r$ *bullet* messages, each of them accompanied by a reply. In addition S_r *alive* or *dead* messages are sent, summing to at most $7S_r$ messages. Due to Lemma 1, there are at most N/S_r^2 candidates in round $r + 1$, for $r \geq 1$. Since $S_{r+1} = S_r^2/2^r$, the processors in round $r + 1$ send together at most $O(N/2^r)$ messages. Summing over all rounds and taking into account the cost of the first round is $S_1 N \in O(N)$ results in $O(N)$ overall communication.

Time complexity. As has been shown above, $S_r \in \Omega(2^{2^{r-1}})$. It follows $O(\log \log N)$ rounds are enough to elect a leader. Let t_r be the time when the first candidate reaches round r . We prove $t_{r+1} \leq t_r + 6$, which is enough to prove the $O(\log \log N)$ time complexity.

Let u be the first processor to reach the round r (at time t_r), v be the first processor in round r to fire *bullet* messages (at time t'_r) and w be the first processor to successfully complete round r (at time t_{r+1}). $t'_r \leq t_r + 2$, because if u survives the target marking phase, it does so at last at the time $t_r + 2$. If u does not survive its marking phase, it has encountered a bullet message at the time at most $t_r + 1$, thus in this case $t'_r \leq t_r + 1$. t_{r+1} can be estimated as $t'_r + 2 + c$, where c is the maximum time spent on waiting for the arrival of *alive* or *dead* message. It easily follows from the algorithm that $c \leq 2$ (time needed for the *OK* or *die!* message to arrive at the candidate and for arrival of *alive* or *dead* message). Summing all together gives the $t_{r+1} \leq t'_r + 4 \leq t_r + 6$.

4 Conclusions

We have shown a $O(N)$ message and $O(\log \log N)$ time election algorithm for asynchronous oriented complete networks. Its message complexity is optimal, and its time complexity is optimal in the class of message optimal algorithms. This closes the gap between the best previously known upper bound of $O(\log N)$ time steps by Singh [16] and the $\Omega(\log \log N)$ lower bound from [10] for asynchronous oriented complete graphs.

As can be seen from the Table 1, the most interesting unsolved problem is the case of unoriented hypercubes, where even the gap between the lower and upper bound for message complexity has not been closed. There is also a space for improvement of the time complexity of message optimal algorithms for wrapped butterflies and CCC.

The complexity of leader election is worth investigating in some other models as well, e.g. in the model of dynamic faults introduced by Santoro and Widmayer in [17].

Acknowledgement. I would like to thank an anonymous referee for numerous comments that helped to improve the presentation of the paper.

References

1. Awerbuch, B.: *Optimal distributed algorithms for minimal weight spanning tree, counting, leader election and related problems*. In Proc. ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 230–240.
2. Burns, J.E.: *A formal model for message passing systems*. Technical Report TR-91, Computer Science Department, Indiana University, Bloomington, Sept. 1980.
3. Dobrev, S. – Ružička, P.: *Linear broadcasting and $N \log \log N$ election in unoriented hypercubes*. In Proc. of SIROCCO'97, Carleton Press, Ascona, Switzerland, 1997, pp. 52–68.
4. Dobrev, S. – Ružička, P.: *Yet Another Modular Technique for Efficient Leader Election*. Proc. of SOFSEM'98, LNCS 1521, Springer-Verlag, 1998, pp. 312–321.
5. Dobrev, S.: *Time and Message Optimal Election in Oriented Hypercubes*. Submitted to SWAT'2000.

6. Flocchini, P. – Mans, B.: *Optimal Elections in Labeled Hypercubes*. Journal of Parallel and Distributed Computing 33 (1), 1996, pp. 76–83.
7. Flocchini, P. – Mans, B. – Santoro, N.: *Sense of direction: definition, properties and classes*. Networks 32(3) 1998, pp. 165–180.
8. Gallager, R. G. – Humblet, P. A. – Spira, P. M.: *A distributed algorithm for minimum-weight spanning trees*. ACM Trans. Programming Languages and Systems 5, 1983, pp. 66–77.
9. Hirschberg, D.S. – Sinclair, J.B.: *Decentralized extrema-finding in circular configurations of processes*. Communication of the ACM 23(11) 1980, pp. 627–628.
10. Israeli, A. – Kranakis, E. – Krizanc, D. – Santoro, N.: *Time-message Trade-offs for the Weak Unison Problem*, Nordic Journal of Computing 4(1997), pp. 317–329.
11. Korach, E. – Moran, S. – Zaks, S.: *Optimal Lower Bounds for Some Distributed Algorithms for a Complete Network of Processors*. TCS 64(1), 1989, pp. 125–132.
12. Loui, M.C. – Matsushita, T.A. – West, D.B.: *Election in complete networks with a sense of direction*. Inf. Proc. Lett. 22, 1986, pp. 185–187. Addendum: Inf. Proc. Lett. 28, 1988, p. 327.
13. Mans, B.: *Optimal Distributed Algorithms in Unlabelled Tori and Chordal Rings*. Journal of Parallel and Distributed Computing 46(1), 1997, pp. 80–90.
14. Peterson, G. L.: *Efficient algorithms for elections in meshes and complete networks*. Technical Report TR140, Dept. of Computer Science, Univ. of Rochester, Rochester, NY 14627, 1985.
15. Singh, G.: *Leader Election in Complete Networks*. SIAM J. COMPUT., 26(3), 1997, pp. 772–785. Preliminary version containing the proof of the lower bound appeared in Proc. of 11th Symposium on Principles of Distributed Computing, 1992,
16. Singh, G.: *Leader Election Using Sense of Direction*. Distributed Computing, 10(3), 1997, pp. 159–165.
17. Santoro, N. – Widmayer, P.: *Distributed function evaluation in presence of transmission faults*, in Proc. of SIGAL'90, Tokyo, 1990; LNCS 450, Springer Verlag, 1990, pp. 358–369.
18. Tel, G.: *Introduction to Distributed Algorithms*. Cambridge University Press, Cambridge, 1994.
19. Tel, G.: *Linear Election in Oriented Hypercubes*. Parallel Processing Letters 5, 1995, pp. 357–366.

Subtractive Reductions and Complete Problems for Counting Complexity Classes^{*}

Arnaud Durand¹, Miki Hermann², and Phokion G. Kolaitis^{3**}

¹ LACL, Dept. of Computer Science, Université Paris 12, 94010 Créteil, France.

durand@univ-paris12.fr

² LORIA (CNRS), BP 239, 54506 Vandœuvre-lès-Nancy, France. hermann@loria.fr

³ Computer Science Dept., University of California, Santa Cruz, CA 95064, U.S.A.

kolaitis@cse.ucsc.edu

Abstract. We introduce and investigate a new type of reductions between counting problems, which we call *subtractive reductions*. We show that the main counting complexity classes $\#P$, $\#NP$, as well as all higher counting complexity classes $\#\Pi_k^P$, $k \geq 2$, are closed under subtractive reductions. We then pursue problems that are complete for these classes via subtractive reductions. We focus on the class $\#NP$ (which is the same as the class $\#coNP$) and show that it contains natural complete problems via subtractive reductions, such as the problem of counting the minimal models of a Boolean formula in conjunctive normal form and the problem of counting the cardinality of the set of minimal solutions of a homogeneous system of linear Diophantine inequalities.

1 Introduction and Summary of Results

Decision problems ask whether a “solution” exists, whereas counting problems ask how many different “solutions” exist. Valiant [Val79a,Val79b] developed a computational complexity theory of counting problems by introducing the class $\#P$ of functions that count the number of accepting paths of nondeterministic polynomial-time Turing machines; thus, $\#P$ captures counting problems whose underlying decision problem (is there a “solution”?) is in NP. Moreover, Valiant demonstrated that $\#P$ contains a wealth of complete problems, i.e., problems in $\#P$ such that every problem in $\#P$ can be reduced to them via a suitable polynomial-time Turing reduction. Clearly, a counting problem is at least as hard as its underlying decision problem. Valiant’s seminal discovery was that there are $\#P$ -complete problems whose underlying decision problem is solvable in polynomial time. The first problem found to exhibit this “easy-to-decide, but hard-to-count” behavior was $\#PERFECT\ MATCHINGS$, which is the problem of counting the number of perfect matchings in a given bipartite graph. Indeed, Valiant [Val79a] showed that $\#PERFECT\ MATCHINGS$ is $\#P$ -complete via polynomial-time 1-Turing reductions, that is, Turing reductions that only allow a single call to an oracle.

^{*} Full version: <http://www.loria.fr/~hermann/publications/mfcs00.ps.gz>

^{**} Research partially supported by NSF Grant CCR-9732041.

In addition to introducing $\#P$, Valiant [Val79a] also developed a machine-based framework for introducing higher counting complexity classes. In this framework, the first class beyond $\#P$ is the class $\#NP$ of functions that count the number of accepting paths of polynomial-time nondeterministic Turing machines with access to NP oracles. More recently, Hemaspaandra and Vollmer [HV95] developed a predicate-based framework for introducing higher counting complexity classes, which subsumes Valiant's framework and makes it possible to introduce other counting classes that draw finer distinctions. In particular, Valiant's class $\#NP$ coincides with the class $\#coNP$ of the Hemaspaandra-Vollmer framework.

As regards complete problems for these higher counting complexity classes, the state of affairs is rather complicated. Toda and Watanabe [TW92] showed if a problem is $\#P$ -hard via polynomial-time 1-Turing reductions, then it is also $\#coNP$ -hard and $\#\Pi_k^P$ -hard, for each $k \geq 2$, where $\#\Pi_k^P$ is the counting version of the class Π_k^P at the k -th level of the polynomial hierarchy PH. This surprising result yields an abundance of problems, such as $\#PERFECT\ MATCHINGS$, that are complete for these higher counting classes. At the same time, it strongly suggests that $\#P$, $\#coNP$, and all other higher counting classes are not closed under polynomial-time 1-Turing reductions. In turn, this means that problems like $\#PERFECT\ MATCHINGS$ do not capture the inherent complexity of the higher counting complexity classes. Needless to say that these classes are closed under *parsimonious* reductions, i.e., polynomial-time reductions that preserve the number of solutions. These reductions, however, also preserve the complexity of the underlying decision problem; thus, they cannot be used to discover the existence of problems that are complete for the higher counting complexity classes and exhibit an “easy-to-decide, but hard-to-count” behavior.

In this paper, we introduce a new type of reductions between counting problems, which we call *subtractive reductions*, since they make it possible to count the number of solutions by first overcounting them and then carefully subtracting any surplus. We make a case that the subtractive reductions are perfectly tailored for the study of $\#coNP$ and of the higher counting complexity classes $\#\Pi_k^P$, $k \geq 2$. To this effect, we first show that each of these higher counting complexity classes is closed under subtractive reductions. We then focus on the class $\#coNP$ and show that it contains natural complete problems via subtractive reductions, such as the problem of counting the minimal models of a Boolean formula in conjunctive normal form and the problem of counting the minimal solutions of a homogeneous system of linear Diophantine inequalities. These two particular counting problems have the added feature that the complexity of their underlying decision problems is lower than Σ_2^P -complete, which is the complexity of the decision problem underlying $\#\Pi_1SAT$, the generic $\#coNP$ -complete problem via parsimonious reductions.

2 Counting Problems and Counting Complexity Classes

A *counting problem* is typically presented using a suitable *witness* function $w: \Sigma^* \rightarrow \mathcal{P}^{<\omega}(\Gamma^*)$, where Σ and Γ are two alphabets, and $\mathcal{P}^{<\omega}(\Gamma^*)$ is the

collection of all finite subsets of Γ^* . Every such witness function gives rise to the following *counting problem*: given a string $x \in \Sigma^*$, find the cardinality $|w(x)|$ of the *witness set* $w(x)$. In the sequel, we will refer to the function $x \mapsto |w(x)|$ as the *counting function* associated with the above counting problem; moreover, we will identify counting problems with their associated counting functions. If a counting problem is described via a witness function w , then the *underlying decision problem for w* asks: given a string x , is the witness set $w(x) \neq \emptyset$?

A *parsimonious reduction* between two counting problems is a polynomial-time many-one reduction that preserves the cardinalities of the witness sets. Valiant [Val79a] introduced the class $\#P$ of counting functions that count the number of accepting paths of nondeterministic polynomial-time Turing machines. The prototypical $\#P$ -complete problem via parsimonious reductions is $\#SAT$: given a Boolean formula φ in conjunctive normal form, find the number of truth assignments to the variables of φ that satisfy φ . Valiant [Val79a, Val79b] also showed that there are $\#P$ -complete problems whose underlying decision problems is solvable in polynomial time; the first problem found to have these properties was $\#PERFECT\ MATCHINGS$. Clearly, unless $P = NP$, such problems cannot be $\#P$ -complete under parsimonious reductions. Instead, $\#PERFECT\ MATCHINGS$ is $\#P$ -complete via *polynomial-time 1-Turing reductions*, where a counting problem v is *polynomial-time 1-Turing reducible* to a counting problem w , if there is a deterministic Turing machine M that computes $|v(x)|$ in polynomial time by making a single call to an oracle that computes $|w(y)|$.

Valiant [Val79a, Val79b] also developed the following framework for introducing higher counting complexity classes: if $\mathcal{C}$ is a class of decision problems, then $\#\mathcal{C}$ is the union $\bigcup_{A \in \mathcal{C}} (\#P)^A$, where $(\#P)^A$ is the collection of all functions that count the accepting paths of nondeterministic polynomial-time Turing machines having A as their oracle. Thus, $\#NP$ is the class of functions that count the number of accepting paths of NP^{NP} machines. Note that $\#\mathcal{C} = \#co\mathcal{C}$ holds for every complexity class $\mathcal{C}$. In particular, $\#NP = \#coNP$; more generally, for every $k \geq 1$, we have that $\#\Sigma_k^P = \#\Pi_k^P$, where Σ_k^P is the k -th level of the polynomial hierarchy PH and $\Pi_k^P = co\Sigma_k^P$ (recall that $\Sigma_1^P = NP$ and $\Pi_1^P = coNP$).

Later on, researchers introduced higher complexity counting classes using a predicate-based framework that focuses on the complexity of membership in the witness sets. Specifically, if $\mathcal{C}$ is a complexity class of decision problems, then Hemaspaandra and Vollmer [HV95] define $\#\cdot\mathcal{C}$ to be the class of all counting problems whose witness function w satisfies the following conditions:

- (1) There is a polynomial $p(n)$ such that for every x and every $y \in w(x)$, we have that $|y| \leq p(|x|)$, where $|x|$ is the length of x and $|y|$ is the length of y ;
- (2) The decision problem “given x and y , is $y \in w(x)$?” is in $\mathcal{C}$.

What is the relationship between counting complexity classes in these two different frameworks? First, it is easy to verify that $\#P = \#\cdot P$. As regards higher counting complexity classes, information about this relationship is provided by Toda’s result [Tod91], which asserts that $\#\cdot\Sigma_k^P \subseteq \#\Sigma_k^P = \#\cdot P^{\Sigma_k^P} = \#\cdot\Pi_k^P$ for every $k \geq 1$ (see also [HV95]). In particular, $\#\cdot NP \subseteq \#NP = \#\cdot P^{NP} = \#\cdot coNP$. This result shows that the predicate-based framework not only subsumes the

machine-based framework, but also makes it possible to make finer distinctions between counting complexity classes that were absent in the machine-based framework. Indeed, for each $k \geq 1$, Valiant's class $\#\Sigma_k^P$ (which is the same as $\#\Pi_k^P$) coincides with $\#\Pi_k^P$. Moreover, the class $\#\Pi_k^P$ appears to be different and, hence, larger than $\#\Sigma_k^P$. In particular, results by Köbler, Schöning, and Torán [KST89] imply that $\#\text{NP} = \#\text{coNP}$ if and only if $\text{NP} = \text{coNP}$.

Do the higher counting complexity classes $\#\Pi_k^P$ (and $\#\Sigma_k^P$) contain natural complete problems and, if so, do some of these problems have an easier underlying decision problem than others? We begin exploring these questions by considering counting problems based on quantified Boolean formulas with a bounded number of quantifier alternations. In what follows, k is a fixed positive integer.

$\#\Pi_k\text{SAT}$

Input: A formula $\varphi(y_1, \dots, y_n) = \forall x_1 \exists x_2 \cdots Q_k x_k \psi(x_1, \dots, x_k, y_1, \dots, y_n)$, where ψ is a Boolean formula, each x_i is a tuple of variables, and each y_j is a variable.

Output: Number of truth assignment to the variables $y_1, \dots, y_n$ that satisfy φ .

Proposition 1. *$\#\Pi_k\text{SAT}$ is $\#\Pi_k^P$ -complete via parsimonious reductions. In addition, if k is odd (even), then the problem remains $\#\Pi_k^P$ -complete when restricted to inputs in which the quantifier-free part is a Boolean formula in disjunctive normal form (respectively, in conjunctive normal form).*

The above result seems to be part of the folklore. A self-contained proof is given in the full paper; it can also be derived from results of Wrathall [Wra76]. The counting problem $\#\Sigma_k\text{SAT}$ is defined in a similar manner and can be shown to be $\#\Sigma_k^P$ -complete via parsimonious reductions.

Note that the decision problem underlying $\#\Pi_k\text{SAT}$ is $\Sigma_{k+1}\text{SAT}$, the prototypical Σ_{k+1}^P -complete problem. Thus, the question becomes: are there any natural $\#\Pi_k^P$ -complete problems whose underlying decision problem is of lower computational complexity (i.e., lower than Σ_{k+1}^P -complete)? Clearly, unless Σ_{k+1}^P collapses to a lower complexity class, no such problem can be $\#\Pi_k^P$ -complete via parsimonious reductions, which means that a broader class of reductions has to be considered. In this vein, Toda and Watanabe [TW92] proved the following surprising result: if a counting problem is $\#\text{P}$ -hard via polynomial-time 1-Turing reductions, then it is also $\#\Pi_k^P$ -complete via the same reductions, for every $k \geq 1$. Consequently, $\#\text{PERFECT MATCHINGS}$ is $\#\Pi_k^P$ -complete via polynomial-time 1-Turing reductions. At first sight, Toda and Watanabe's theorem [TW92] can be interpreted as providing an abundance of $\#\Pi_k^P$ -complete problems such that their underlying decision problem is of low complexity. A moment's reflection, however, reveals that this theorem provides strong evidence that $\#\text{P}$, $\#\text{coNP}$, and all other higher counting complexity classes $\#\Pi_k^P$, $k \geq 2$, are *not* closed under polynomial-time 1-Turing reduction. Indeed, if $\#\Sigma_k^P$ is closed under 1-Turing reductions, then $\#\Sigma_k^P = \#\text{PH}$, which in turn implies a collapse of the polynomial hierarchy to Σ_k^P . Similarly, if $\#\Pi_k^P$ is closed under 1-Turing reductions, then $\#\Pi_k^P = \#\text{PH}$, implying a collapse of the polynomial hierarchy to $\text{UP}^{\Sigma_k^P}$. Moreover, this theorem implies that polynomial-time 1-Turing reductions

cannot help us discover complete problems that embody the inherent difficulty of each counting complexity class $\#\Pi_k^P$, $k \geq 1$, and allow us to draw meaningful distinctions between these classes. Consequently, the challenge is to discover a different class of reductions having the following two crucial properties: (1) each class $\#\Pi_k^P$, $k \geq 1$, is closed under these reductions; (2) each class $\#\Pi_k^P$, $k \geq 1$, contains natural problems that are complete for the class via these reductions. In what follows, we take the first steps towards confronting this challenge.

3 Subtractive Reductions

Researchers in structural complexity theory have extensively investigated various *closure properties* of $\#P$ and of certain other counting complexity classes (see [HO92, OH93]). For instance, it is well known and easy to prove that $\#P$ is closed under both addition and multiplication.¹ In contrast, $\#P$ does *not* appear to be closed under subtraction, since Ogiwara and Hemachandra [OH93] have shown that $\#P$ is closed under subtraction if and only if the class PP of problems solvable in probabilistic polynomial time coincides with the class UP of problems solvable by an unambiguous Turing machine in polynomial time, which is considered an unlikely eventuality. This state of affairs suggests that considerable care has to be exercised in defining reductions under which $\#P$ and other higher counting complexity classes are closed.

In this section, we introduce the class of *subtractive reductions* that first overcount and then carefully subtract any surplus items. We begin by defining some auxiliary concepts and establishing notation.

Let D be a nonempty set. Intuitively, a *multiset* on D is a collection of elements of D in which elements may have multiple occurrences. More formally, a *multiset* M on D can be viewed as a function $M: D \rightarrow \mathbb{N}$ that assigns to each element $x \in D$ the number $M(x)$ of the occurrences of x in M . The multisets on D can be equipped with the operations of *union* and *difference* as follows.

Let A and B be two multisets on D . The *union* of A and B is the multiset $A \oplus B$ such that $(A \oplus B)(x) = A(x) + B(x)$ for every $x \in D$. The *difference* of A and B is the multiset $A \ominus B$ such that $(A \ominus B)(x) = \max(A(x) - B(x), 0)$ for every $x \in D$. We say that A is *contained* in B , and write $A \subseteq B$, if $A(x) \leq B(x)$ for every $x \in D$. Note that if $B \subseteq A$, then $(A \ominus B)(x) = A(x) - B(x)$ holds for all $x \in D$. Finally, if $A_1, \dots, A_n$ are multisets, then we write $\bigoplus_{i=1}^n A_i$ to denote the union $A_1 \oplus \dots \oplus A_n$.

Let Σ, Γ be two alphabets and let $R \subseteq \Sigma^* \times \Gamma^*$ be a binary relation between strings such that, for each $x \in \Sigma^*$, the set $R(x) = \{y \in \Gamma^* \mid R(x, y)\}$ is finite. We write $\# \cdot R$ to denote the following counting problem: given a string $x \in \Sigma^*$, find the cardinality $|R(x)|$ of the witness set $R(x)$ associated with x . It is easy to see that every counting problem is of the form $\# \cdot R$ for some R .

Definition 2. Let Σ, Γ be two alphabets and let A and B be two binary relations between strings from Σ and Γ . We say that the counting problem $\# \cdot A$ reduces to

¹ Apparently, K. Regan was the first to observe this closure property of $\#P$, see [HO92].

the counting problem $\# \cdot B$ via a subtractive reduction, and write $\# \cdot A \leq_s \# \cdot B$, if there exist a positive integer n and polynomial-time computable functions f_i and g_i , $i = 1, \dots, n$, such that for every string $x \in \Sigma^*$:

- $\bigoplus_{i=1}^n B(f_i(x)) \subseteq \bigoplus_{i=1}^n B(g_i(x))$;
- $|A(x)| = \sum_{i=1}^n |B(g_i(x))| - \sum_{i=1}^n |B(f_i(x))|$.

Parsimonious reductions constitute a very special case of subtractive reductions. We note that in the sequel we will produce subtractive reductions between counting problems $\# \cdot A$ and $\# \cdot B$ that involve only one pair of functions f and g such that $B(f(x)) \subseteq B(g(x))$ and $|A(x)| = |B(g(x))| - |B(f(x))|$. The full generality of subtractive reductions is needed, however, to establish that they possess the following desirable property (a proof can be found in the full paper).

Theorem 3. *Reducibility via subtractive reductions is a transitive relation.*

Next we establish the main result of this section; it asserts that Valiant's counting complexity classes are closed under subtractive reductions.

Theorem 4. *$\#P$ and all higher counting complexity class $\# \cdot \Pi_k^P = \# \Sigma_k^P$, $k \geq 1$, are closed under subtractive reductions.*

Proof. (Sketch) Let k be a fixed positive integer. In what follows, we sketch the proof that the class $\# \cdot \Pi_k^P$ is closed under subtractive reductions; the proof for $\#P$ requires only minor modifications. Recall that Toda [Tod91] showed that $\# \cdot \Pi_k^P = \# \Sigma_k^P = \# \cdot P^{\Sigma_k^P}$. Let $\# \cdot A$ and $\# \cdot B$ be two counting problems such that $\# \cdot B \in \# \cdot \Pi_k^P$ and $\# \cdot A$ reduces to $\# \cdot B$ via subtractive reduction. We will show that $\# \cdot A$ belongs to $\# \cdot \Pi_k^P$ by constructing a predicate A' in $P^{\Sigma_k^P}$ such that where f_i and g_i , $1 \leq i \leq n$, are the polynomial-time computable function in the subtractive reduction of $\# \cdot A$ to $\# \cdot B$. The elements of the predicate A' will be pairs of strings (x, y') such that $y' = f_1(x) * \dots * f_n(x) * g_1(x) * \dots * g_n(x) * y * z$, where z is an integer ranging from 1 to the number b of occurrences of y in the multiset $\bigoplus_i^n B(g_i(x)) \ominus \bigoplus_i^n B(f_i(x))$, and $*$ is just a delimiter symbol.

The predicate A' is constructed as follows. A pair (x, y') belongs to A' if and only if (x, y') is accepted by the following algorithm:

1. extract $f_1(x), \dots, f_n(x), g_1(x), \dots, g_n(x), y$, and z from y' ;
2. find the number c_g of pairs $(g_i(x), y)$, $1 \leq i \leq n$, that belong to B ;
3. find the number c_f of pairs $(f_i(x), y)$, $1 \leq i \leq n$, that belong to B ;
4. check that $z \leq c_g - c_f$.

Step 4 ensures that, for every y , there are as many accepted strings y' as the number of occurrences of y in the multiset $\bigoplus_i^n B(g_i(x)) \ominus \bigoplus_i^n B(f_i(x))$. Therefore, the number of pairs (x, y') accepted by A' is equal to the number of pairs $(x, -)$ accepted by A . Step 1 can be carried out in polynomial time. For each pair in Step 2, the test is in Π_k^P ; moreover, c_g is bounded by the fixed number n of the functions g_i . Hence, Step 2 is in $P^{\Sigma_k^P}$. For each pair in Step 3, the test is in Σ_k^P ; moreover, c_f is also bounded by n . Hence, as above, Step 3 is in $P^{\Sigma_k^P}$. Step 4 can be carried out in polynomial time. Thus, the predicate A' is in $P^{\Sigma_k^P}$. $\square$

In view of Theorem 4, it is natural to ask whether the classes $\#\cdot\Sigma_k^P$, $k \geq 1$, introduced by Hemaspaandra and Vollmer [HV95], are also closed under subtractive reductions. We provide the evidence that this is *not* the case. For this, observe that $\#\Pi_k\text{SAT}$, the generic complete problem for $\#\cdot\Pi_k^P$, can easily be reduced to $\#\Sigma_k\text{SAT}$, the generic complete problem for $\#\cdot\Sigma_k^P$, via a subtractive reduction. Consequently, if $\#\cdot\Sigma_k^P$ were closed under subtractive reductions, then $\#\cdot\Pi_k^P$ would collapse to $\#\cdot\Sigma_k^P$, which is generally considered as highly unlikely.

Let $\varphi(y_1, \dots, y_n)$ be any Π_k -formula $\forall x_1 \exists x_2 \dots Q_k x_k \phi(x_1, \dots, x_k, y_1, \dots, y_n)$. Let $\bar{\varphi}(y_1, \dots, y_n)$ be the Σ_k formula that is equivalent to $\neg\varphi$ and is obtained from φ by propagating the negation symbol through the quantifiers and applying de Morgan laws to the quantifier-free part of φ . Let $\psi(y_1, \dots, y_n)$ be the tautology $y_1 \vee \neg y_1 \vee y_2 \vee \neg y_2 \vee \dots \vee y_n \vee \neg y_n$. It is obvious that every satisfying truth assignment of $\bar{\varphi}$ is a satisfying truth assignment of ψ and that $\#(\varphi) = \#(\psi) - \#(\bar{\varphi})$, where $\#(\varphi)$ denotes the number of satisfying truth assignments of φ (and similarly for ψ and $\bar{\varphi}$). Consequently, the polynomial-time computable functions $f_1(\varphi) = \bar{\varphi}$ and $g_1(\varphi) = \psi$ constitute a subtractive reduction of $\#\Pi_k\text{SAT}$ to $\#\Sigma_k\text{SAT}$.

Observe that the preceding argument can also be applied to a Boolean formula φ in conjunctive normal form (i.e., assume $k = 0$) to produce a subtractive reduction of $\#\text{SAT}$ to $\#\text{DNF}$, where $\#\text{DNF}$ is the problem of counting the satisfying truth assignments to a Boolean formula in disjunctive normal form. Hence, $\#\text{DNF}$ is $\#\text{P}$ -complete via subtractive reductions. Observe that $\#\text{DNF}$ cannot be $\#\text{P}$ -complete via parsimonious reductions, since its underlying decision problem is easily solvable in polynomial time.

As stated earlier, $\#\text{PERFECT MATCHINGS}$ is $\#\text{P}$ -complete via polynomial-time 1-Turing reductions. It is an interesting open problem to determine whether $\#\text{PERFECT MATCHINGS}$ is also $\#\text{P}$ -complete via subtractive reductions.

4 $\#\cdot\text{coNP}$ -Complete Problems via Subtractive Reductions

Recall that $\#\cdot\text{coNP}$ is the first higher counting complexity class in Valiant's framework, because $\#\cdot\text{coNP} = \#\text{NP}$. Moreover, $\#\cdot\text{coNP}$ is quite robust, since, as shown by Toda [Tod91], $\#\cdot\text{coNP} = \#\text{NP} = \#\cdot\text{P}^{\text{NP}}$. In this section, we establish that $\#\cdot\text{coNP}$ contains certain natural counting problems that possess the following two properties: (1) they are $\#\cdot\text{coNP}$ -complete via subtractive reductions; (2) their underlying decision problems has complexity lower than Σ_2^P -complete, which is the complexity of the decision problem underlying $\#\Pi_1\text{SAT}$, the generic $\#\cdot\text{coNP}$ -complete problem via parsimonious reductions.

Circumscription is a well-developed formalism of common-sense reasoning introduced by McCarthy [McC80] and extensively studied by the artificial intelligence community. The key idea behind circumscription is that one is interested in the *minimal models* of formulas, since they are the ones that have as few "exceptions" as possible and, therefore, embody common sense. In the context of Boolean logic, circumscription amounts to the study of satisfying assignments

of Boolean formulas that are *minimal* with respect to the *pointwise partial order* on truth assignments. More precisely, if $s = (s_1, \dots, s_n)$ and $s' = (s'_1, \dots, s'_n)$ are two elements of $\{0, 1\}^n$, then we write $s < s'$ to denote that $s \neq s'$ and $s_i \leq s'_i$ holds for every $i \leq n$. Let $\varphi(x_1, \dots, x_n)$ be a Boolean formula having $x_1, \dots, x_n$ as its variables and let $s \in \{0, 1\}^n$ be a truth assignment. We say that s is a *minimal model* of φ if s is a satisfying truth assignment of φ and there is no satisfying truth assignment s' of φ such that $s < s'$. This concept gives rise to the following natural counting problem.

#CIRCUMSCRIPTION

Input: A Boolean formula $\varphi(x_1, \dots, x_n)$ in conjunctive normal form.

Output: Number of minimal models of $\varphi(x_1, \dots, x_n)$.

The underlying decision problem for #CIRCUMSCRIPTION is NP-complete, since a Boolean formula has a minimal model if and only if it is satisfiable. Thus, its complexity is lower than Σ_2^P -complete.

Theorem 5. #CIRCUMSCRIPTION is #coNP-complete via subtractive reductions.

Proof. (Hint) The problem belongs to #coNP, since testing whether a given truth assignment is a minimal model of a given formula is in coNP. For the lower bound, we construct a subtractive reduction of # Π_1 SAT to #CIRCUMSCRIPTION. In what follows, we write $A(F)$ to denote the set of all satisfying assignments of a Π_1 -formula F ; we also write $B(\psi)$ to denote the set of all minimal models of a Boolean formula ψ . Let $F(x) = \forall y \phi(x, y)$ be a Π_1 -formula, where $\phi(x, y)$ is a Boolean formula in DNF, and $x = (x_1, \dots, x_n)$, $y = (y_1, \dots, y_m)$ are tuples of Boolean variables. Let $x' = (x'_1, \dots, x'_n)$ be a tuple of new Boolean variables, let z be a single new Boolean variable, let $P(x, x')$ be the formula $(x_1 \equiv \neg x'_1) \wedge \dots \wedge (x_n \equiv \neg x'_n)$, let $Q(y)$ be the formula $y_1 \wedge \dots \wedge y_m$, and, finally, let $F'(x, x', y, z)$ be the formula $P(x, x') \wedge (z \rightarrow Q(y)) \wedge (\phi(x, y) \rightarrow z)$.

There is a polynomial-time computable function g such that, given a Π_1 -formula F as above, it returns a Boolean formula $g(F)$ in CNF that is logically equivalent to the formula $F'(x, x', y, z)$ (this is so, because $\phi(x, y)$ is in DNF). Now let $F''(x, x', y, z)$ be the formula $F'(x, x', y, z) \wedge (z \rightarrow \neg Q(y))$ and let f be a polynomial-time computable function such that, given a Π_1 -formula F as above, it returns a Boolean formula $f(F)$ in CNF that is logically equivalent to the formula $F''(x, x', y, z)$. It can be shown that $|A(F)| = |B(F')| - |B(F'')|$, which establishes that the polynomial-time computable functions f and g constitute a subtractive reduction of # Π_1 SAT to #CIRCUMSCRIPTION. $\square$

An immediate consequence of Theorems 4 and 5 is that #coNP = #P if and only if #CIRCUMSCRIPTION is in #P.

We now move from counting problems in Boolean logic to counting problems in integer linear programming. A *system of linear Diophantine inequalities over the nonnegative integers* is a system of the form $S: Ax \leq b$, where A is an integer matrix, b is an integer vector, and we are interested in the nonnegative integer solutions of this system. If b is the zero-vector $(0, \dots, 0)$, then we say that the system is *homogeneous*. A nonnegative integer solution s of S is *minimal* if there

is no nonnegative solution s' of S such that $s' < s$ in the pointwise partial order on integer vectors. It is well known that the set of all minimal solutions plays an important role in analyzing the space of all nonnegative integer solutions of linear Diophantine systems (see Schrijver [Sch86]). Clearly, every homogeneous system has $(0, \dots, 0)$ as a trivial minimal solution. Here, we are interested in counting the number of nontrivial minimal solutions of homogeneous systems.

#HOMOGENEOUS MIN SOL

Input: A homogeneous system $S: Ax \leq 0$ of linear Diophantine inequalities.

Output: Number of nontrivial minimal solutions of S .

Note that the underlying decision problem of #HOM MIN SOL amounts to whether a given homogeneous system of linear Diophantine inequalities has a nonnegative integer solution other than the trivial solution $(0, \dots, 0)$. It is easy to show that this problem is solvable in polynomial time, since it can be reduced to LINEAR PROGRAMMING. In contrast, counting the number of nontrivial minimal solutions turns out to be a hard problem.

Theorem 6. #HOMOGENEOUS MIN SOL is #coNP-complete via subtractive reductions.

Proof. (*Hint*) The problem is in #coNP, because deciding membership in the witness sets is in coNP; indeed, the size of minimal solutions is bounded by a polynomial in the size of the system (see Corollary 17.1b in [Sch86, page 239]). The lower bound is established through a sequence of subtractive reductions. First, #CIRCUMSCRIPTION can be reduced to #SATISFIABLE CIRC, the restriction of #CIRCUMSCRIPTION to satisfiable Boolean formulas. In turn, this problem has a subtractive reduction to #SATISFIABLE MIN SOL, which asks for the number of minimal solutions of a system $S: Ax \leq b$ of linear Diophantine inequalities having at least one nonnegative integer solutions (details of these two reductions can be found in the full paper). Finally, #SATISFIABLE MIN SOL has a subtractive reduction to #HOMOGENEOUS MIN SOL, which we now outline.

Let $S: Ax \leq b$ be a system of linear Diophantine inequalities with at least one nonnegative integer solution and such that A is $k \times n$ integer matrix. First construct the system $S': Ax - b\bar{y} \leq 0, 2z - t = y, x_i \leq y, x_i \geq y - t$, where $\bar{y} = (y, \dots, y)$ is a vector of length k having the same variable y in each coordinate, and z and t are additional new variables. After this, construct the system $S'' = S' \cup \{x_1 = \dots = x_n = y\}$.

Let $A(S)$ be the set of minimal solutions of the system S , and let $B(S')$ and $B(S'')$ be the sets of nontrivial minimal solutions of S' and S'' , respectively. It can be shown that $B(S'') \subseteq B(S')$ and that $|A(S)| = |B(S')| - |B(S'')|$. This establishes that the polynomial-time computable functions $f(S) = S'$ and $g(S) = S''$ constitute a subtractive reduction of #SATISFIABLE MIN SOL to #HOMOGENEOUS MIN SOL. $\square$

The previous theorem implies the following collapse: #coNP = #P if and only if #HOMOGENEOUS MIN SOL is in #P. To the best of our knowledge, the above result provides the first example of a counting problem whose underlying decision problem is solvable in polynomial time, but the counting problem itself is not in #P, unless higher counting complexity classes collapse to #P.

5 Concluding Remarks

In his influential paper [Val79b], Valiant asserted that “*The completeness class for $\#P$ appears to be rivalled only by that for NP in relevance to naturally occurring computational problems.*” The passage of time and the subsequent research in this area certainly proved this to be the case. We believe that the results reported here suggest that also $\#\text{-coNP}$ contains complete problems of computational significance and that subtractive reductions are the right tool for investigating $\#\text{-coNP}$ and identifying other natural $\#\text{-coNP}$ -complete problems. The next challenge in this vein is to determine whether $\#\text{HILBERT}$ is $\#\text{-coNP}$ -complete via subtractive reductions. $\#\text{HILBERT}$ is the problem of computing the cardinality of the *Hilbert basis* of a homogeneous system $S: Ax = 0$ of linear Diophantine equations, i.e., counting the number of nontrivial minimal solutions of such a system. We note that this counting problem was first studied by Hermann, Juban and Kolaitis [HJK99], where it was shown to be a member of $\#\text{-coNP}$ and also to be $\#P$ -hard under polynomial-time 1-Turing reductions.

References

- [HJK99] M. Hermann, L. Juban, and P. G. Kolaitis. On the complexity of counting the Hilbert basis of a linear Diophantine system. In *Proc. 6th LPAR*, volume 1705 of *LNCS (in AI)*, pages 13–32, September 1999. Springer.
- [HO92] L. A. Hemachandra and M. Ogiwara. Is $\#P$ closed under subtraction? *Bulletin of the EATCS*, 46:107–122, February 1992.
- [HV95] L. A. Hemaspaandra and H. Vollmer. The satanic notations: Counting classes beyond $\#P$ and other definitional adventures. *SIGACT News*, 26(1):2–13, 1995.
- [KST89] J. Köbler, U. Schöning, and J. Torán. On counting and approximation. *Acta Informatica*, 26(4):363–379, 1989.
- [McC80] J. McCarthy. Circumscription — A form of non-monotonic reasoning. *Artificial Intelligence*, 13(1-2):27–39, 1980.
- [OH93] M. Ogiwara and L. A. Hemachandra. A complexity theory for feasible closure properties. *Journal of Computer and System Science*, 46(3):295–325, 1993.
- [Sch86] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1986.
- [Tod91] S. Toda. *Computational complexity of counting complexity classes*. PhD thesis, Tokyo Institute of Technology, Dept. of Computer Science, Tokyo, 1991.
- [TW92] S. Toda and O. Watanabe. Polynomial-time 1-Turing reductions from $\#PH$ to $\#P$. *Theoretical Computer Science*, 100(1):205–221, 1992.
- [Val79a] L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979.
- [Val79b] L. G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.
- [Wra76] C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

On the Autoreducibility of Random Sequences

Todd Ebert¹ and Heribert Vollmer²

¹ DoCoMo Communications Laboratories USA, 250 Cambridge Ave.,
Palo Alto, CA 94306.

e-mail address: toddebert@dcl.docomo-usa.com

² Theoretische Informatik, Universität Würzburg, Am Hubland,
97074 Würzburg, Germany.

e-mail address: vollmer@informatik.uni-wuerzburg.de

Abstract. A language $A \subseteq \{0,1\}^*$ is called i.o. autoreducible if A is Turing-reducible to itself via a machine M such that, for infinitely many input words w , M does not query its oracle A about w . We examine the question if algorithmically random languages in the sense of Martin-Löf are i.o. autoreducible. We obtain the somewhat counterintuitive result that every algorithmically random language is polynomial-time i.o. autoreducible where the autoreducing machine poses its queries in a “quasi-nonadaptive” way; however, if in the above definition the “infinitely many” is replaced by “almost all,” then every algorithmically random language is not autoreducible in this stronger sense. Further results obtained give upper and lower bounds on the number of queries of the autoreducing machine M and the number of inputs w for which M does not query the oracle about w .

1 Introduction

This paper arose from an investigation of the problem of deducing a property of a random binary sequence when some of the bits of the sequence upon which the property depends are not known. This occurs quite often in practice when, due to time and other resource constraints, a decision is made using only partial information. Not surprisingly, this consideration is closely related to complexity theory, since a decision must be made before a limited resource such as time has been exhausted.

To introduce the question we study in this paper, let us consider the following puzzle.

The N Prisoners Puzzle

A group of N prisoners ($N > 1$) were awaiting parole. The parole committee split on whether or not to grant all of them freedom. When splits occurred, the warden would decide a prisoner’s fate by tossing a coin on his desk, covering it with his hand, and having the prisoner guess the side facing up. If the prisoner guessed correctly, he was set free; otherwise he would serve more time. Since they

were tried and convicted of the same crime and had displayed equal amounts of good behavior, he used a slightly different approach with the N prisoners. He said to them, “I’ve tossed N coins onto my desk, one for each of you. Each of you will enter my office one at a time and have the opportunity to guess if your coin, which will be covered by my hand, is showing heads or tails. The other prisoners’ coins will be visible to you. You may not ask me any questions whatsoever. Furthermore, after leaving my office, you will be unable to communicate anything to the other prisoners. Not all of you have to guess, but at least one of you must guess. If all the guesses made are correct, then all of you will go free. But if one of you guesses incorrectly, then *all* of you will remain in prison.”

At first sight, since every prisoner is unable to observe any event or obtain any information which the outcome of its coin toss is probabilistically dependent on, one expects that the prisoners should have no more than a 50% chance for freedom. However, since they converse before the game, we demonstrate how collaboration can benefit the entire group.

As an example, take $N = 3$ and suppose the prisoners agree to the following guessing strategy. Upon entering the warden’s office, each prisoner observes the two unconcealed coins on the warden’s desk. If both coins have the same side facing up, then the prisoner guesses his coin is showing the opposite side. However, if the coins have opposite sides facing up, then the prisoner leaves the office without venturing a guess. To compute the chances for freedom under this strategy, two possibilities must be considered. The first case occurs when all three coins have the same side facing up. In this case, each prisoner will incorrectly guess. The second case is two of the three coins show the same side facing up. Here exactly one prisoner will venture a correct guess, and freedom is won. Hence the probability for freedom using the above strategy is $1 - \frac{1}{8} - \frac{1}{8} = \frac{3}{4}$.

The General Problem

We study a more general problem in relativized complexity theory in which an oracle machine M with oracle A has unlimited time and space resources; yet in deciding if word x belongs to the oracle property L^A , machine M may not have access to some of the bits of A . We are particularly interested in the case when $L^A = A$. More precisely, we allow the deterministic oracle machine to use A as oracle set, but for infinitely many x , M may not query the oracle about the bit of A which encodes the answer to whether or not x belongs to A . However, in all cases M has to compute the correct result.

If we imagine that A is generated by the independent tosses of a fair coin, and that the sequence of independent random variables $\{\xi_0, \xi_1, \dots\}$ represents the coin-toss outcomes, then the *a posteriori* probability of guessing the outcome of ξ_j given knowledge of $\xi_{i_1}, \dots, \xi_{i_k}$ equals the *a priori* probability of guessing ξ_j provided $j \notin \{i_1, \dots, i_k\}$. Thus it seems natural to regard the information obtained from observing ξ_i as unhelpful for guessing ξ_j , if $i \neq j$. Hence, at first glance it would seem certain that no oracle machine M with limitations as described above could possibly decide A , since for infinitely many x , M would have to guess the membership of x , and would fail half the time.

On the other hand, by the weak law of large numbers, for all $\epsilon > 0$, $\left\| \frac{\sum_{i=0}^n \xi_i}{n} - .5 \right\| < \epsilon$ with arbitrarily high probability when n is sufficiently large. Thus, independent random events considered collectively possess different statistical properties with asymptotically high probability, and the crux of the following investigation rests on determining degrees to which statistical properties may be used to occasionally determine the outcome of a random variable by observing the outcomes of other random variables independent of the one in question. For the scenario described above, we show how error-correcting codes can be used as the means for allowing oracle machines to decide random languages, despite their querying limitations.

Results

To accomplish this we use infinite Martin-Löf random sequences for our random sequence model, and the known concept of autoreducibility for capturing the idea of using a language A as oracle set in order to decide A , yet not being able to query A about the bit to be decided.

Indeed, a sequence A is called *autoreducible* if A is Turing reducible to itself via an oracle machine M which almost always decides a bit of A without querying A about the bit. If the machine, after one round of nonadaptive queries, is able to either decide its input or recognize that it has to query its oracle about its input, then we say that A is *quasi-tt-autoreducible*. Moreover, if M infinitely often decides a bit of A without querying the oracle about the bit, then A is called *i.o. autoreducible*. Autoreducibility was first studied by Trakhtenbrot [20], see [21, pp. 483ff], and has since then been used in quite different contexts in recursion theory as well as complexity theory.

For reasons described above, we originally surmised that no Martin-Löf random sequence is i.o. autoreducible. In fact it follows, using Kolmogorov complexity [12], that no Martin-Löf random sequence is i.o. tt-autoreducible. (A sequence that is i.o. tt-autoreducible can be compressed by one bit infinitely often; but no random sequence has this property.) Surprisingly, however, we prove that *every* Martin-Löf random sequence is i.o. quasi-tt-autoreducible, even via an autoreducing machine that runs in polynomial time. This result seems paradoxical in that the machine which witnesses the reducibility has the task of infinitely often guessing a bit of a random sequence, and guessing correctly each time despite a 50% chance of error for each guess. We then show how this result strongly relies on a Turing machine's ability to make an unlimited number of queries on each input. This is accomplished by proving no random sequence is i.o. quasi-btt-autoreducible.

Finally, we introduce the notion of autoreducibility with rate $t(n)$ as a gauge of how often an oracle machine can guess the bits of a random sequence. In words, a sequence is autoreducible with rate $t(n)$ iff it is i.o. autoreducible, and the n th guess by the witnessing oracle machine is for bit j_n , where $j_n \leq t(n)$. We prove that every Martin-Löf random sequence is autoreducible with rate $O(n^2 \log^2 n)$, and that this bound is near optimal in the sense that no Martin-Löf random sequence is autoreducible with rate $O(n)$.

2 Preliminaries and Some Basic Facts

2.1 Words, Languages, and Machines

To begin, $\mathbb{N}$ represents the set of nonnegative integers. We consider words over the binary alphabet $\{0, 1\}$, and we assume the words are lexicographically ordered, where s_i , $i \geq 0$, denotes the i th word in the ordering. The set of infinite binary sequences is denoted by $\{0, 1\}^\infty$. The set of infinite sequences which have a prefix x in common is denoted by $x\{0, 1\}^\infty$, called the *cylinder set generated by x* . There is a natural way of forming a one-to-one correspondence between languages over the alphabet $\{0, 1\}$ and infinite binary sequences: $A \subseteq \{0, 1\}^*$ is identified with

$$c_A(s_0)c_A(s_1)\cdots,$$

where c_A denotes the characteristic function of A . Furthermore, $A[0 \dots n]$ denotes the first $n + 1$ bits of the sequence, while $A(n)$ denotes the n th bit.

We are particularly interested in oracle machines, and we denote $L(M, A)$ as the language decided by Turing machine M using A as oracle set. Every input s_i to an oracle machine induces a binary *query tree* T_{s_i} whose nodes are labeled with query words, and whose leaves are labeled with either YES or NO. The computation of M on input s_i proceeds down the branch corresponding to the answers provided by the oracle. For example, if the root node is labeled with s_1 , then M will proceed right if s_1 belongs to oracle set A , and left otherwise. s_i will belong to $L(M, A)$ iff the computation reaches a leaf labeled “YES”. $M^A(x)$ will denote the computation of M on input x using oracle A . The computation is said to *halt* iff M traverses a finite branch of the query tree.

2.2 Martin-Löf Random Sequences

Now assume an effective enumeration $W_0, W_1, \dots$ of all recursively enumerable sets of finite binary words. We may assume that each W_i is a prefix-free set, i.e. if x is a prefix of y and $x, y \in W_i$, then $x = y$. In fact, there is an effective way to convert every r.e. set W to a prefix-free set W' such that $W\{0, 1\}^\infty = W'\{0, 1\}^\infty$. With this fact in hand, we define a probability measure $\Pr$ on the set of r.e. sets. Indeed, given prefix-free r.e. set W ,

$$\Pr(W) = \sum_{x \in W} \frac{1}{2^{|x|}}.$$

A set $\mathcal{B} \subset \{0, 1\}^\infty$ is called a *constructive null set* iff there exists total recursive $g: \mathbb{N} \rightarrow \mathbb{N}$ such that $\mathcal{B} \subseteq \bigcap_{i=0}^\infty W_{g(i)}\{0, 1\}^\infty$ and $\Pr(W_{g(i)}) < \frac{1}{2^i}$. Here, $\{W_{g(i)}\}_{i \in \mathbb{N}}$ is called a *constructive null cover* for $\mathcal{B}$.

The class **NULL** is defined as the union of all constructive null sets, and **RAND** is defined as $\mathbf{RAND} = \{0, 1\}^\infty - \mathbf{NULL}$.

$A \in \{0, 1\}^\infty$ is called a *Martin-Löf random sequence* iff $A \in \mathbf{RAND}$. Since the set of recursive functions is countable, it follows that $\Pr(\mathbf{RAND}) = 1$, while $\Pr(\mathbf{NULL}) = 0$, since $\mathbf{NULL}$ is contained in a countable union of sets having measure zero.

The above definition of randomness is attributed to Per Martin-Löf [16], and is widely accepted as having captured the essence of randomness among infinite binary sequences.

In working with Martin-Löf random sequences it helps to view them as the result of independently tossing an infinite sequence of fair coins; this is true in the sense that, whenever such a sequence is formed, the result must be Martin-Löf random, since $\mathbf{NULL}$ itself is a constructive null set (of impossible outcomes). Henceforth, by “random sequence” we will mean a Martin-Löf random sequence.

2.3 Autoreducibility

A sequence $A \in \{0, 1\}^\infty$ is called *autoreducible* iff there exists a deterministic oracle machine M such that $A \leq_T A$ via M , and for almost every x , $x \notin Q(M, A, x)$, where $Q(M, A, x)$ is the set of query words occurring during the computation of $M^A(x)$. Moreover, if “for almost every” is replaced by “infinitely often”, then A is called *i.o. autoreducible*.

In a similar way we define *tt-autoreducible*, *btt-autoreducible*, *i.o. tt-autoreducible*, and *i.o. btt-autoreducible*. For example, A is *i.o. tt-autoreducible* iff A is *i.o. autoreducible* via some machine M for which query tree T_{s_i} is finite for every $i \in \mathbb{N}$. If $f: \{0, 1\}^* \rightarrow \mathbb{N}$ is such that $f(s_i)$ is an upper bound for the number of nodes in T_x , then we say that A is *i.o. f-tt-autoreducible*. Also, A is called *i.o. btt-autoreducible* iff A is *i.o. k-tt-autoreducible* for some constant $k \in \mathbb{N}$. From these definitions it immediately follows that every recursive sequence is 0-tt-autoreducible. Slightly more subtle is the observation that every r.e. sequence is *i.o. 0-tt-autoreducible*. This easily follows from the fact that every infinite r.e. sequence contains an infinite recursive subsequence. On the other hand, it can easily be shown using a standard diagonalization argument that there are not *i.o. autoreducible* sequences.

Finally we say that a sequence A is *quasi-tt-autoreducible*, if A is autoreducible via a machine M that for every input x first, in a nonadaptive manner, poses a number of queries to A , all of which are different from x , and then determines if it can answer if $x \in A$ itself or it has to ask A about x . Only in this latter case, a second round of queries, only consisting of x , is necessary. Hence, these reductions are very near to tt-reductions. For a function f , we use *f-quasi-tt-autoreducible*, if the number of queries in the first round is bounded by f evaluated for the input length. If f is constant, we use the term *quasi-btt-autoreducible*.

Now consider infinite random sequences. Intuitively, no random sequence is autoreducible since this would require a deterministic oracle machine to correctly guess every bit of the random sequence. This will follow from a stronger result to be proved later. It would also seem plausible that no random sequence is *i.o. autoreducible*, since correctly guessing infinitely often without error seems just as implausible. However, we prove a surprising result: *every* random sequence is

i.o. autoreducible. Indeed, for any random sequence A there is an oracle machine M^A which infinitely often decides a bit of A by correctly guessing after querying A about bits that come before and after the bit being decided. We will make use of the following well-known result, see [19, p. 255]:

Lemma 1 (Borel-Cantelli). *Let $\mathcal{X}$ be a probability space with probability measure $\Pr$, and events $E_0, E_1, \dots$ such that*

$$\sum_{i=0}^{\infty} \Pr(E_i) < \infty.$$

Then

$$\Pr\left(\bigcap_{i=0}^{\infty} \bigcup_{j=i}^{\infty} E_j\right) = 0.$$

In other words, with probability 0, an infinite number of E_i will be satisfied.

2.4 Error-Correcting Codes

A *perfect one-error-correcting code* is a set $\mathcal{C}_n \subseteq \{0, 1\}^n$ such that, for every $x \in \{0, 1\}^n$, either $x \in \mathcal{C}_n$ or

$$\exists! y \in \mathcal{C}_n : d(x, y) = 1,$$

where $d(x, y)$ denotes the Hamming distance between x and y .

It is well known (see, e.g., [11]) that such codes exist for $n + 1 = 2^k$, for all $k \geq 0$. This seems intuitively plausible since each code word forms the center of a unit sphere consisting of $n + 1 = 2^k$ words, in which the other n words are error words. Since the spheres are mutually disjoint, 2^{n-k} equals the number of spheres, and hence code words. Furthermore, $\mathcal{C}_n$ is a binary $(n - k)$ -dimensional vector subspace of $\{0, 1\}^n$, and $x \in \mathcal{C}_n$ iff $xH^t = 0$; where H is a $k \times n$ *parity check matrix* for $\mathcal{C}_n$ whose n columns are simply the binary representations of the numbers 1 through n . Thus we have proved the following theorem.

Theorem 2. *If $n + 1 = 2^k$, then there exists a perfect one-error-correcting code $\mathcal{C}_n$ of size 2^{n-k} . Moreover, $\bigcup_{n \in \mathbb{N}} \mathcal{C}_n$ is polynomial-time decidable.*

3 Results

Theorem 3. *Every random sequence is i.o. quasi-tt-autoreducible.*

Proof. Define a sequence $a_0 = \frac{1}{2^{n_0}}, a_1 = \frac{1}{2^{n_1}}, \dots$ so that $\sum_k a_k$ converges. Partition the natural numbers so that the k th partition has size $2^{n_k} - 1$. Notice that, for every $k \in \mathbb{N}$, there is a perfect one-error-correcting code $\mathcal{C}_k$ contained in the set $\{0, 1\}^{m_k}$, where $m_k = 2^{n_k} - 1$. We may further assume that the set of codes $\mathcal{C}_0, \mathcal{C}_1, \dots$ is recursively enumerable.

Next define an oracle machine M in a manner in which M attempts to be a witness for i.o. autoreducibility for an arbitrary sequence A . The strategy M uses involves viewing A as the concatenation of words $v_0, v_1, \dots$, where $|v_k| = 2^{n_k} - 1$. Furthermore, M assumes that $v_k \notin \mathcal{C}_k$, for every $k \in \mathbb{N}$.

With this in mind, on input s_j , M first determines which partition j belongs to. Assuming that j belongs to the k th partition, M first queries the oracle about all words s_l such that l belongs to the k th partition and $l \neq j$. Thus M determines two words x and y , and knows that the portion of A corresponding to the k th partition is either $w_0 = x0y$ or $w_1 = x1y$, where the uncertainty lies in the j th bit of A , and the size of both w_0 and w_1 is $2^{n_k} - 1$. If neither w_0 nor w_1 belong to $\mathcal{C}_k$, then M queries the oracle about s_j and accepts s_j iff the answer is YES. On the other hand, if $w_0 \in \mathcal{C}_k$, then $w_1 \notin \mathcal{C}_k$, since two code words cannot be a unit distance apart. In this case M accepts s_j , since this is consistent with a one in the j th position, and M is assuming that v_k is an error word. Likewise, M would reject s_j if $w_1 \in \mathcal{C}_k$.

The correctness of this strategy rests on the following

Claim: If $A = v_0v_1v_2\dots$, where v_k is an error word for code $\mathcal{C}_k$, then A is i.o. quasi-tt-autoreducible via M .

To prove this, consider v_k for arbitrary $k \geq 0$. If v_k is an error word for $\mathcal{C}_k$, then there exists a unique code word w_k such that $d(v_k, w_k) = 1$. Assume that the v_k and w_k differ at bit j , and suppose this bit encodes membership of s_{j_k} in A . Then M will decide bit j_k without querying the oracle about that bit. Indeed, M may query the bits occurring before and after j_k to determine that the portion of A corresponding to partition k is either v_k or w_k . It correctly chooses v_k since it is assuming that A is comprised of the concatenation of error words. Thus, M may decide bit j_k without querying about j_k . For all other bits occurring in the k th partition, M may follow the same procedure, but will find that v_k and w_k are both error words, and hence cannot distinguish between the two. In this case M concedes by querying the oracle about the bit to be decided. Therefore, A is i.o. quasi-tt-autoreducible via M , and the claim is proved.

To finish the proof of the theorem, simply note that a language A will not be quasi-tt-autoreducible via M iff there exists a k such that v_k is a code word. However, if there are only a finite number of such k , then A is i.o. quasi-tt-autoreducible via a machine M' which patches the finite number of mistakes made by M . Moreover, since

$$\Pr(v_k \text{ is a code word}) = \frac{1}{2^{n_k}} = a_k,$$

and $\sum_k a_k$ converges, a finite number of mistakes will occur with probability one (Borel-Cantelli). Finally, notice that the set of sequences $\mathcal{B}$ for which an infinite number of mistakes occur forms a constructive null set. Indeed, letting E_k denote the set of sequences for which v_k is a code word, we have

$$\mathcal{B} = \bigcap_{i=0}^{\infty} \left(\bigcup_{j=i}^{\infty} E_j \right),$$

and clearly there exists a recursive function $g: \mathbb{N} \rightarrow \mathbb{N}$ such that the expression in parentheses may be expressed as $W_{g(i)}\{0, 1\}^\infty$. Hence the set is constructive null and every random sequence must be i.o. quasi-tt-autoreducible. $\square$

We note in passing that, by choosing the a_k sufficiently small, we may have M witness i.o. autoreducibility for a measure of sequences arbitrarily approaching 1.

Returning to the N prisoners puzzle introduced in Sect. 1, from what we have done above, it is obvious that the prisoners can optimize their chance for freedom by assuming the coin sequence forms an error word of an error-correcting code in some dimension $\leq N$. For example, if $N = 7$, then the set of code words forms a 4-dimensional vector subspace; thus there are $128 - 16 = 112$ error words in the space $\{0, 1\}^7$, and with probability $\frac{112}{128} = .875$ the prisoners will be set free.

The oracle machine constructed in the proof of Theorem 3 has the feature of making an unlimited number of queries to the oracle before deciding membership of a word. Furthermore, if $r(x)$ is the number of queries needed to decide x , then $r(x)$ is a step-like function which grows without bound. Conversely, suppose $r: \{0, 1\}^* \rightarrow \mathbb{N}$ is an unbounded, increasing recursive function. Using the error-correcting codes as in Theorem 3 we choose a set of bits $\{i_1, \dots, i_n\}$ and assume that the binary vector $A(i_1) \cdots A(i_n)$ is an error-word for some error-correcting code. If $n + 1$ is a power of 2 then we will be correct with probability $n/(n + 1)$. From the definition of the autoreduction machine in the proof of Theorem 3, it is clear that for every k , $1 \leq k \leq n$, the query tree $T(s_{i_k})$ will have n nodes. Since r is unbounded, we may choose $i_1, \dots, i_k$ in such a way that $r(s_{i_k}) \geq n$ for $1 \leq k \leq n$. Thus, we proved:

Theorem 4. *Let $r: \{0, 1\}^* \rightarrow \mathbb{N}$ be an unbounded, increasing recursive function. Then every random sequence is i.o. r -autoreducible.*

The just given theorem readily allows us to conclude that the number of queries made by an i.o. autoreducing machine can be made polynomial in the length of the input. Since deciding if a word belongs to an error-correcting code is in polynomial-time we obtain:

Theorem 5. *Every random sequence is polynomial-time i.o. autoreducible.*

As was observed by Wolfgang Merkle [17], the proof of the above results does not really need the full power of constructive null-covers. Specifically it can be shown, using the above idea of stretching the intervals that contain code words far enough, that every set which is not polynomial-time i.o. autoreducible can be covered by a polynomial-time computable martingale. This implies that every p-random set (in the sense of Lutz [14,15]) is polynomial-time i.o. autoreducible.

The next theorem shows that having $r(x)$ grow without bound is a necessary condition for random sequences to be i.o. autoreducible. For a proof, which has to be omitted here, we refer the reader to [9].

Theorem 6. *No random sequence is i.o. quasi-btt-autoreducible.*

From Theorem 3 we know that every random sequence is i.o. autoreducible. An interesting problem involves finding lower bounds to the frequency at which a deterministic oracle machine decides the bit of a random sequence without querying about that bit.

Suppose A is i.o. autoreducible via M , and suppose $i_0, i_1, \dots$ is the increasing sequence of bit locations that M guesses (i.e., in the computation on input s_{i_k} , M does not query the oracle about s_{i_k} for $k = 0, 1, \dots$), for every $n \in \mathbb{N}$. Then A is called *autoreducible with rate $t(n)$* iff $i_n \leq t(n)$, for every $n \in \mathbb{N}$.

Thus, autoreducibility with rate $t(n)$ measures the rate at which M guesses bits of A . This yields a rate of $\frac{n}{t(n)}$ guesses per bits decided. So the question arises as to what is the highest rate a machine may achieve with a random sequence. We may find a partial answer to this in the proof of Theorem 3, by noting that $t(n)$ depends on the length of the codewords used for each of the error-correcting codes. In this case the codeword length for the k th code is given by

$$L_k = 2^{n_k} - 1 = \frac{1}{a_k} - 1.$$

Thus, in choosing a slowly converging sequence such as $a_k = \frac{1}{k \log^2 k}$, and letting n_k be the least integer such that 2^{n_k} exceeds $\frac{1}{a_k}$, we see that the n th guess will occur before the first $t(n)$ bits have been decided, where

$$t(n) = \sum_{k=1}^n 2k \log^2 k = O(n^2 \log^2 n).$$

And we have proved the first statement of the following theorem; a proof of the second statement, proceeding analogous to the one of Theorem 6, can again be found in [9].

Theorem 7. 1. *Every random sequence is quasi-tt-autoreducible with a rate of $O(n^2 \log^2 n)$.*
2. *No random sequence is quasi-tt-autoreducible with a rate of $O(n)$.*

Considering (everywhere) autoreducibility we obtain the following corollary.

Corollary 8. *No random sequence is autoreducible.*

In the resource-bounded setting, the question if p-random sets can be (everywhere) polynomial-time autoreducible was examined in [7]. In that paper, autoreducible sets were highlighted for “testing the power of resource-bounded measure.” The preceding corollary, restricted to polynomial-time autoreducibility, already appeared there.

Acknowledgments. We are very grateful to Wolfgang Merkle (Universität Heidelberg) for a number of helpful discussions. We especially thank Ken Rose (UC Santa Barbara) for his enlightening discussions on error-correcting codes. We also acknowledge helpful hints from Klaus W. Wagner (Universität Würzburg), Dieter van Melkebeek (University of Chicago), Charles Akemann (UC Santa Barbara) and an anonymous referee.

References

1. Alon N., Spencer J. *The Probabilistic Method*. John Wiley and Sons, New York, 1992.
2. Bennett C., Gill J. *Relative to a Random Oracle A*, $P^A \neq NP^A \neq coNP^A$ With Probability 1. *SIAM Journal on Computing*, 10, 96-113, 1981.
3. Blahut R. *Theory and Practice of Error Control Codes*. Addison Wesley, Reading, Ma, 1984.
4. Book R. *On Languages Reducible to Algorithmically Random Languages*. *SIAM Journal on Computing*, Vol. 23, No. 6, 1275-1282, 1984.
5. Book R., Lutz J., Wagner K. W. *An Observation on Probability Versus Randomness*. *Math. Systems Theory*, 27, 201-209, 1994.
6. Brualdi R. *Introductory Combinatorics*. North-Holland, NY, 1988.
7. Buhrman H., van Melkebeek D., Regan K.W., Sivakumar D., Strauss M., A generalization of resource-bounded measure, with an application to the BPP vs. EXP problem. Manuscript, 1999. Preliminary version appeared in Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science, pp. 161-171, 1998, and as University of Chicago, Department of Computer Science, Technical Report TR-97-04, May 1997.
8. Cutland N. *Introduction to Computability*. Cambridge Univ. Press, 1992.
9. Ebert T. *Applications of Recursive Operators to Randomness and Complexity*. Ph.D. Thesis, University of California at Santa Barbara, 1998.
10. Gács P. *Every Sequence is Reducible to a Random One*. *Information and Control*, 70, 186-192, 1986.
11. Kim K.H., Roush F.W. *Applied Abstract Algebra*. John Wiley and Sons, New York, 1983.
12. Li M., Vitanyi P. *An Introduction to Kolmogorov Complexity and its Applications*. Springer-Verlag, New York, 1993.
13. Lutz J. *König's Lemma, Randomness, and the Arithmetical Hierarchy*. Unpublished Lecture Notes, Iowa St. Univ., 1993.
14. Lutz J. *A Pseudorandom Oracle Characterization of BPP*. *SIAM Journal on Computing*, 22(5):1075-1086, 1993.
15. Lutz J. *The Quantitative Structure of Exponential Time*. In: Hemaspaandra L., Selman A., editors, *Complexity Theory Restrospective II*, Springer Verlag, 1997, 225-260.
16. Martin-Löf P. *On the Definition of Random Sequences*. *Information and Control*, 9, 602-619, 1966.
17. Wolfgang Merkle. Personal communication, December 1999.
18. Rogers H. *Theory of Recursive Functions and Effective Computability*. MIT Press, Cambridge, 1992.
19. Shiryayev A.N. *Probability*. Springer-Verlag, New York, 1995.
20. Trakhtenbrot, B.A. *On Autoreducibility*. *Soviet Math dokl.*, 11:814-817, 1970.
21. Wagner K. W., Wechsung G. *Computational Complexity*. Deutscher Verlag der Wissenschaften, Berlin, 1986.

Iteration Theories of Boolean Functions

Zoltán Ésik*

University of Szeged
Department of Computer Science
P.O.B. 652, 6701 Szeged, Hungary
esik@inf.u-szeged.hu

1 Introduction

A systematic study of the fixed point (or dagger) operation in Lawvere algebraic theories was initiated by Elgot and the ADJ group. Their work led to the introduction of iteration theories in 1980, which capture the equational properties of fixed points in the models proposed by Elgot and the ADJ group. The book [2] and the survey paper [3] provide ample evidence that the axioms of iteration theories have a general scope and constitute a complete description of the equational properties of the fixed point operation.

The lattice of all theories (or clones) of boolean functions was described by Emil Post in 1920, but no proof was published until 1941, see [8]. In this paper we prove that all iteration theories of boolean functions equipped with a pointwise dagger operation consist of monotonic functions, and that the dagger operation is either the least, or the greatest fixed point operation. Along the way of obtaining this result, we relate the parameter identity, an equation that holds in all Conway and iteration theories, to the pointwise property of the dagger operation, and to the extension of the dagger operation to the theory obtained by adjoining all constants. We also exhibit an iteration theory of boolean functions with a non-pointwise dagger, an iteration theory of monotonic functions on the three-element lattice which has a pointwise dagger but such that the dagger operation is not one of the extremal fixed point operations, and a pointwise iteration theory on the three-element set such that there is no cpo structure which would make all functions of the theory monotonic.

2 Theories and Iteration Theories

A *theory of functions on a set* A is a category T whose objects are the integers $n \geq 0$ and whose morphisms $n \rightarrow m$ are functions $A^n \rightarrow A^m$ including the projections $\text{pr}_i^{A^n} : A^n \rightarrow A$, $i \in \mathbf{n} = \{1, \dots, n\}$, $n \geq 0$. Composition is function composition denoted $\cdot$. Moreover, T is closed under target tupling: if $f_1, \dots, f_m : A^n \rightarrow A$ are in T , where $m, n \geq 0$, then so is the function $f = \langle f_1, \dots, f_m \rangle : A^n \rightarrow A^m$.

* Partially supported by grant no. FKFP 247/1999 from the Ministry of Education of Hungary and grant no. T22423 from the National Foundation of Hungary for Scientific Research.

$A^n \rightarrow A^m$ defined by $f(x_1, \dots, x_n) = (f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n))$. In particular, T contains the identity functions $\mathbf{1}_{A^n} = \langle \text{pr}_1^A, \dots, \text{pr}_n^A \rangle$ and the functions $!_{A^n} : A^n \rightarrow A^0$. For example, when A is a poset, the *monotonic* functions on A form a theory denoted $\mathbf{Mon}_A$. A *theory* is any category whose objects are the natural numbers which is isomorphic to a theory of functions under an isomorphism which maps each object n to itself. We assume that each theory comes with a specified family of projections and a target tupling operation. In any theory, we will denote the projection morphisms as pr_i^n , the identity morphisms as $\mathbf{1}_n$ and the morphisms $n \rightarrow 0$ as $!_n$. A *theory morphism* between theories is a functor that preserves the objects and the projections (hence the tupling operation).

Suppose that T and T' are theories such that each hom-set of T is included in the corresponding hom-set of T' . We call T' a *subtheory* of T if the inclusion $T' \rightarrow T$ is a theory morphism. Thus, T' is a subcategory of T and has the same projection morphisms as T .

A *preiteration theory* is a theory T equipped with a *dagger operation*

$$f : n + p \rightarrow n \mapsto f^\dagger : p \rightarrow n.$$

The restriction of this operation to *scalar morphisms*, i.e., to morphisms $1 + p \rightarrow 1$, is called the *scalar dagger operation*. A *preiteration theory morphism* $T \rightarrow T'$ is a theory morphism which also preserves the dagger operation. *Preiteration subtheories* are defined in the expected way.

A basic example of a preiteration theory is the theory $\mathbf{Mon}_L^0$ of *monotonic functions* on a *complete lattice* L equipped with the *least fixed point operation* defined as follows. For each monotonic function $f : L^{n+p} \rightarrow L^n$ and $a \in L^p$, $f^\dagger(a)$ is the least fixed point of the map $L^n \rightarrow L^n$, $x \mapsto f(x, a)$. Thus,

$$f^\dagger(a) = f(f^\dagger(a), a), \quad (1)$$

and for all $b \in L^n$, $b = f(b, a) \Rightarrow f^\dagger(a) \leq b$. (Since L is a complete lattice, so is L^n equipped with the pointwise partial order.) Moreover, $f^\dagger$ is a monotonic function $L^p \rightarrow L^n$. The same facts are known to hold for monotonic functions over any *cpo*. (Recall that a *cpo* is a non-empty partially ordered set P such that each directed subset of P , including the empty set, has a supremum.) Thus, each *cpo* P gives rise to a “pointwise” preiteration theory $\mathbf{Mon}_P^0$. For later use we note that when P is finite with k elements, then the least fixed point can be obtained as $f^\dagger(a) = f^{n^k}(\perp, \dots, \perp, a)$, where $\perp$ denotes the least element of P , and the *powers* of f are defined by $f^0 = \text{pr}_1^{n,p}$, $f^{m+1} = f \cdot \langle f^m, \text{pr}_2^{n,p} \rangle$, $m \geq 0$. (For unexplained notions and notation, see the Appendix.) Dually, if L is a complete lattice, one may also define the preiteration theory $\mathbf{Mon}_L^1$ of monotonic functions on L equipped with the *greatest fixed point operation*.

The preiteration theories $\mathbf{Mon}_L^0$ and $\mathbf{Mon}_L^1$ satisfy a number of non-trivial equations involving the dagger operation. First, by (1) above, *Elgot’s fixed point equation*

$$f^\dagger = f \cdot \langle f^\dagger, \mathbf{1}_p \rangle, \quad f : n + p \rightarrow n \quad (2)$$

holds in all such theories. Other well-known identities are the *parameter* (3), *pairing* (4), *composition* (5), *double dagger* (6) and *commutative identities* (7). (The pairing identity appears in [1,5] and the composition and double dagger identities in [7].)

$$(f \cdot (\mathbf{1}_n \times g))^\dagger = f^\dagger \cdot g, \quad f : n + p \rightarrow n, \quad g : q \rightarrow p. \quad (3)$$

The special case when $n = 1$ is called the *scalar parameter identity*.

$$\langle f, g \rangle^\dagger = \langle f^\dagger \cdot \langle h^\dagger, \mathbf{1}_p \rangle, h^\dagger \rangle, \quad (4)$$

for all $f : n + m + p \rightarrow n$ and $g : n + m + p \rightarrow m$, where $h : m + p \rightarrow m$ is defined by $h = g \cdot \langle f^\dagger, \mathbf{1}_{m+p} \rangle$. The special case that $m = 1$ is called the *scalar pairing identity*.

$$(f \cdot \langle g, \text{pr}_2^{n,p} \rangle)^\dagger = f \cdot \langle (g \cdot \langle f, \text{pr}_2^{m,p} \rangle)^\dagger, \mathbf{1}_p \rangle, \quad (5)$$

all $f : m + p \rightarrow n$ and $g : n + p \rightarrow m$. The special case that $n = m = 1$ is called the *scalar composition identity*.

$$f^{\dagger\dagger} = (f \cdot (\langle \mathbf{1}_n, \mathbf{1}_n \rangle \times \mathbf{1}_p))^\dagger, \quad f : n + n + p \rightarrow n. \quad (6)$$

When $n = 1$, this is called the *scalar double dagger identity*.

$$\langle f \cdot (\rho_1 \times \mathbf{1}_p), \dots, f \cdot (\rho_n \times \mathbf{1}_p) \rangle^\dagger = \langle f^\dagger, \dots, f^\dagger \rangle, \quad (7)$$

for all $f : n + p \rightarrow 1$ and all base morphisms $\rho_i : n \rightarrow n$, $i \in \mathbf{n}$. (For this form of the commutative identity as well as a refinement, see [6].)

Definition 1. A **Conway theory** is a preiteration theory satisfying the *parameter* (3), *composition* (5) and *double dagger* (6) identities. An **iteration theory** is a Conway theory satisfying the *commutative identities* (7). A **morphism** of Conway or iteration theories is a preiteration theory morphism.

It is known that the pairing and fixed point identities hold in all Conway theories. An alternative axiomatization of Conway theories is given by the scalar versions of the *parameter*, *composition*, *double dagger* and *pairing* identities. The notion of iteration theories is justified by the following result:

Theorem 1. [2] An equation holds in all preiteration theories $\mathbf{Mon}_L^0$, where L is any complete lattice, or in all preiteration theories $\mathbf{Mon}_P^0$, where P is a cpo, iff it holds in iteration theories.

Of course, an equation holds in all preiteration theories $\mathbf{Mon}_L^0$ iff it holds in all preiteration theories $\mathbf{Mon}_L^1$.

3 Pointwise Dagger

A *preiteration theory of functions on a set A* is a preiteration theory T which is a subtheory of the theory $\mathbf{F}_A$ of all functions over A .

Definition 2. Suppose that T is a preiteration theory of functions on a set A . We say that the **dagger operation is pointwise**, or that T is a **pointwise preiteration theory of functions**, if for all $f : A^{n+p} \rightarrow A^n$ and $g : A^{n+q} \rightarrow A^n$ in T , and for all $a \in A^p$, $b \in A^q$,

$$[\forall x \in A^n f(x, a) = g(x, b)] \Rightarrow f^\dagger(a) = g^\dagger(b). \quad (8)$$

Clearly, $\mathbf{Mon}_L^0$ and $\mathbf{Mon}_L^1$ are pointwise preiteration theories, in fact, pointwise iteration theories defined below. The same holds for the preiteration theories $\mathbf{Mon}_P$, where P is any cpo. The following facts are clear.

Proposition 1. Any pointwise preiteration theory of functions satisfies the parameter identity.

Proposition 2. Suppose that T is a preiteration theory of functions over a set A . If T contains all constants $A^0 \rightarrow A$, then the parameter identity holds in T iff the dagger operation is pointwise.

When T is a theory of functions over a set A , let $T(A)$ denote the smallest subtheory of $\mathbf{F}_A$ containing T and all of the constants $A^0 \rightarrow A$. It is easy to see that $T(A)$ consists of all functions of the form $f \cdot (\mathbf{1}_{A^n} \times a)$, where $f : A^{n+p} \rightarrow A^n$ in T and $a \in A^p$, which is conveniently identified with the corresponding function $A^0 \rightarrow A^p$, also denoted a .

Theorem 2. Suppose that T is a preiteration theory of functions. Then the dagger operation can be extended from T to the whole of $T(A)$ such that the parameter identity holds in $T(A)$ iff the dagger operation on T is pointwise. Moreover, the extension is unique.

Proof. If the extended dagger operation on $T(A)$ satisfies the parameter identity, then by Proposition 2, the extended dagger operation is pointwise. Hence the dagger operation on T is also pointwise. Suppose now that the dagger operation defined on T is pointwise. Given a function $f : A^{n+p} \rightarrow A^n$ in $T(A)$, write f as $f = g \cdot (\mathbf{1}_{A^{n+p}} \times a)$ in T , where $g : A^{n+p+q} \rightarrow A^n$ and $a \in A^q$. We define $f^* = g^\dagger \cdot (\mathbf{1}_{A^p} \times a)$, where of course $g^\dagger$ is taken in T . That the new operation $*$ is well-defined follows from the assumption that the dagger operation is pointwise on T . It is now clear that $f^* = f^\dagger$ whenever f is in T . Moreover, since the dagger operation on T is pointwise, so is the extended dagger operation. By Proposition 2, the parameter identity holds in $T(A)$. $\square$

For a proof of the following lemma, see [2].

Lemma 1. Suppose that T is a preiteration theory satisfying the scalar pairing identity. Then the parameter identity holds in T iff the scalar parameter identity does.

Let T be a preiteration theory of functions over A . We say that the *scalar dagger operation is pointwise* if condition (8) holds when $n = 1$, i.e., when f is any function $A^{1+p} \rightarrow A$ in T .

Lemma 2. *Let T be a preiteration theory of functions on A . Suppose that T satisfies the scalar pairing identity. Then the dagger operation is pointwise iff so is the scalar dagger operation.*

Proof. Suppose that the scalar dagger operation on T is pointwise. Then, similarly as in the proof of Theorem 2, the scalar dagger operation can be extended to all functions $f : A^{1+p} \rightarrow A$ in $T(A)$ such that the scalar parameter identity holds. Next, using the scalar pairing identity, the dagger operation can be extended to all functions $f : A^{n+p} \rightarrow A^n$ in $T(A)$. Since the scalar pairing identity holds in T , the dagger operation on $T(A)$ extends the dagger operation given on T . Moreover, by Lemma 1, the parameter identity holds in $T(A)$. Thus, by Proposition 2, the dagger operation on $T(A)$, and henceforth on T , is pointwise. $\square$

Suppose that T is a pointwise preiteration theory of functions on A . By Theorem 2, there is a unique way to extend the dagger operation to $T(A)$ such that the parameter identity holds. We can show that when T is a Conway theory, or iteration theory, then so is $T(A)$. Actually this follows from Theorem 3 which gives a sufficient condition ensuring that $T(A)$ be a conservative extension of T . Before presenting this general result, we need some definitions.

A *preiteration theory term* is any well-formed sorted term constructed from sorted variables $f : m \rightarrow n$ and constants $\text{pr}_i^{m_1, m_2}$, $i \in \mathbf{2}$, by using the operations of composition, tupling and iteration. Below we will assume that for each triple (m, n, p) , the map $f \mapsto f_p$ is an injective function from the set of variables of sort $m \rightarrow n$ to the set of variables of sort $m + p \rightarrow n$. When $p = 0$, we choose $f_p = f$, for all variables f . The defining identities of Conway and iteration theories may be given in “parameterless form”, and each such equation induces an equation for any p -tuple of the parameters. This process is captured by the following (rather technical) definition.

Definition 3. *Suppose that t is a term of sort $m \rightarrow n$ and p is a non-negative integer. The **translated term** t_p of sort $m + p \rightarrow n$ is defined by induction on the structure of t .*

1. If $t = f$ is a variable, then $t_p = f_p$.
2. If $t = \text{pr}_i^{m_1, m_2}$, then $t_p = \text{pr}_i^{m_1, m_2, p}$.
3. If $t = t' \cdot t''$, where $t' : k \rightarrow n$ and $t'' : m \rightarrow k$, then $t_p = t'_p \cdot \langle t''_p, \text{pr}_2^{m, p} \rangle$.
4. If $t = \langle t', t'' \rangle$, then $t_p = \langle t'_p, t''_p \rangle$.
5. If $t = u^\dagger$, then $t_p = (u_p)^\dagger$.

For a preiteration theory T of functions on a set A and evaluation κ of the sorted variables $f : r \rightarrow s$ by functions $f\kappa : A^r \rightarrow A^s$ in $T(A)$, each term t of sort $m \rightarrow n$ determines a function $t\kappa : A^m \rightarrow A^n$ in $T(A)$, defined in the usual way. Given a subset F of the variables, an evaluation κ of the variables in $T(A)$, an integer $p \geq 0$ and $a \in A^p$, suppose that there is an evaluation $\bar{\kappa}$ of the variables in

T such that for all $f : r \rightarrow s$ in F , $(f\kappa)(x) = (f_p\bar{\kappa})(x, a)$, all $x \in A^r$, i.e., $f\kappa = f_p\bar{\kappa} \cdot (\mathbf{1}_{A^r} \times a)$. For any term t , below we will write just t for $t\kappa$ and t_p for $t_p\bar{\kappa}$. By induction on the structure of t we can prove:

Lemma 3. *If T has a pointwise dagger, then for all terms $t : m \rightarrow n$ whose variables are in F , $t(x) = t_p(x, a)$, for all $x \in A^m$.*

Lemma 4. *If T is any theory of functions on a set A and $f_i : A^{m_i} \rightarrow A^{n_i} \in T(A)$, $i \in \mathbf{k}$, then there exist $p \geq 0$, $a \in A^p$ and functions $f'_i : A^{m_i+p} \rightarrow A^{n_i}$ in T such that for all $i \in \mathbf{k}$, $f_i = f'_i \cdot (\mathbf{1}_{A^{m_i}} \times a)$, i.e., $f_i(x) = f'_i(x, a)$, for all $x \in A^{m_i}$.*

Theorem 3. *Suppose that T is a pointwise preiteration theory of functions on a set A and t and t' are terms of sort $m \rightarrow n$. If $t_p = t'_p$ holds in T for all $p \geq 0$, then the equation $t = t'$ holds in $T(A)$.*

Proof. We need to show that for all evaluations κ of the morphism variables in $T(A)$, and for all $x \in A^m$, it holds that $(t\kappa)(x) = (t'\kappa)(x)$. Let F denote the finite set of variables occurring in the terms t and t' . By Lemma 4, there exist an integer $p \geq 0$ and a vector $a \in A^p$ such that each $f\kappa$ with $f : r \rightarrow s$ in F can be written in the form $\bar{f} \cdot (\mathbf{1}_{A^r} \times a)$ for some $\bar{f} : A^{r+p} \rightarrow A^s$ in T . Let $\bar{\kappa}$ be an evaluation of the variables in T such that $f_p\bar{\kappa} = \bar{f}$, for each $f \in F$. By Lemma 4 we have $(t\kappa)(x) = (t_p\bar{\kappa})(x, a)$ and $(t'\kappa)(x) = (t'_p\bar{\kappa})(x, a)$, for all $x \in A^m$. But the equation $t_p = t'_p$ holds in T , so that $(t_p\bar{\kappa})(x, a) = (t'_p\bar{\kappa})(x, a)$ and $(t\kappa)(x) = (t'\kappa)(x)$, $x \in A^m$. Since this holds for all κ , we have proved that T satisfies the equation $t = t'$. $\square$

Remark 1. Under the assumptions of Theorem 3, $t_p = t'_p$ holds in $T(A)$ for all $p \geq 0$.

Corollary 1. *If T is a Conway theory (iteration theory, respectively) of functions on A which is equipped with a pointwise dagger, then $T(A)$ is a Conway theory (iteration theory, respectively).*

Proof. The defining identities of Conway and iteration theories are closed under translation. $\square$

4 Pointwise Iteration Theories of Boolean Functions

Definition 4. *Let T be a pointwise preiteration theory of functions over a set A . If T is also an iteration theory (Conway theory, respectively), then we call T a pointwise iteration theory (pointwise Conway theory, respectively).*

We have already noted that $\mathbf{Mon}_L^0$ and $\mathbf{Mon}_L^1$ are pointwise iteration theories, for every complete lattice L . In particular, $\mathbf{Mon}_2^0$ and $\mathbf{Mon}_2^1$ are also pointwise iteration theories. In this section we describe all of the pointwise iteration theories of boolean functions, i.e., the pointwise iteration theories on the set $\mathbf{2}$. We will show that any pointwise Conway theory of boolean functions is a pointwise iteration theory.

Lemma 5. *Suppose that T is a preiteration theory of boolean functions satisfying the scalar fixed point identity. Then any function in T is monotonic.*

Proof. Assume, towards a contradiction, that T contains a non-monotonic function. Then T contains a non-monotonic function $f : \mathbf{2}^{1+p} \rightarrow \mathbf{2}$. In fact, by an appropriate permutation of the arguments of f , we may assume that f is non-monotonic in its first argument. Thus, there exists $a \in \mathbf{2}^p$ such that $f(0, a) = 1$ and $f(1, a) = 0$. But then there is no solution to the fixed point equation $x = f(x, a)$, so that the scalar fixed point identity fails. $\square$

Lemma 6. *Let T be a pointwise Conway theory of boolean functions such that $\mathbf{1}_2^\dagger = 0$, the constant function $\mathbf{2}^0 \rightarrow \mathbf{2}$ with value 0. Then for each $f : \mathbf{2}^{n+p} \rightarrow \mathbf{2}^n \in T$ and $a \in \mathbf{2}^p$, $f^\dagger(a)$ is the least solution of the fixed point equation*

$$x = f(x, a). \quad (9)$$

Dually, if $\mathbf{1}_2^\dagger = 1$, then for each f and a as above, $f^\dagger(a)$ is the greatest solution of (9).

Proof. We only consider the case that $\mathbf{1}_2^\dagger = 0$. By the pairing identity (4), we only need to show that $f^\dagger(a)$ is the least solution of (9) when $n = 1$, i.e., when $f : \mathbf{2}^{1+p} \rightarrow \mathbf{2}$ in T . But in that case, by Lemma 5, the function $f_a = f \cdot (\mathbf{1}_2 \times a)$, $x \mapsto f(x, a)$ is either constant or the identity function $\mathbf{1}_2$. In the first case, $f^\dagger(a)$ is the unique solution of (9). In the second case, since the dagger operation is pointwise, $f^\dagger(a) = 0$ is the least solution. $\square$

Theorem 4. 1. *Suppose that T is a preiteration theory of boolean functions.*

Then T is a pointwise Conway theory iff T is a preiteration subtheory of $\mathbf{Mon}_2^0$ or $\mathbf{Mon}_2^1$. In either case, T is a pointwise iteration theory.

2. *A theory T of boolean functions can be turned into an iteration theory iff T is contained in $\mathbf{Mon}_2$ and contains a constant $\mathbf{2}^0 \rightarrow \mathbf{2}$.*

- a) *If $0 \in T$, then there is a unique way to turn T into a pointwise iteration theory such that $\mathbf{1}_2^\dagger = 0$: the dagger operation is the least fixed point operation.*
- b) *Dually, if $1 \in T$, then there is a unique way to turn T into a pointwise iteration theory such that $\mathbf{1}_2^\dagger = 1$: the dagger operation is the greatest fixed point operation.*
- c) *If both constants 0 and 1 are in T , then there are exactly two iteration theory structures on T , the dagger operation is either the least or the greatest fixed point operation, so that the dagger is pointwise.*

Proof. The necessity of the first claim follows from Lemmas 5 and 6. The sufficiency is immediate, since $\mathbf{Mon}_2^0$ and $\mathbf{Mon}_2^1$ are pointwise iteration theories, and any preiteration subtheory of a pointwise iteration theory of is an iteration theory with a pointwise dagger. For the second claim, it has been shown in Lemma 5 above that if a theory T of boolean functions can be turned into an Conway theory, then T is a subtheory of $\mathbf{Mon}_2$. Moreover, T contains the

constant $\mathbf{1}_2^\dagger$. If $0 \in T$, then for each $f : \mathbf{2}^{n+p} \rightarrow \mathbf{2}^n$, the function $g = f^{2^n} \cdot \langle 0, \dots, 0, \mathbf{1}_{2^p} \rangle : \mathbf{2}^p \rightarrow \mathbf{2}^n$ is in T . But for any $a \in \mathbf{2}^p$, $g(a)$ is the least solution of the fixed point equation $x = f(x, a)$. Thus, by letting $f^\dagger = g$, the dagger operation is well-defined, moreover, by Theorem 1, T is a (pointwise) iteration theory. The uniqueness follows from Lemma 6. If both 0 and 1 are in T , then by Proposition 2, any Conway theory structure on T has a pointwise dagger operation. $\square$

On the set $\mathbf{2}$, there is, up to isomorphism, a single cpo structure, given by $0 \leq 1$. The set $A = \mathbf{3}$ can be turned into a cpo essentially in two different ways. One either considers the usual total order, which determines a lattice, or the partial order such that $0 \leq 1$, $0 \leq 2$ and 1 and 2 are incomparable. Taking the monotonic functions with respect to the second partial order, equipped with the least fixed point operation, the resulting pointwise iteration theory contains functions which are not monotonic with respect to the total order. More interestingly, there exists a pointwise iteration theory T on $A = \mathbf{3}$ with the following property: there is no cpo structure on A such that all functions in T would be monotonic. Let T be the subtheory of $\mathbf{F}_A$ generated by the constants $A^0 \rightarrow A$ and the functions $f : A \rightarrow A$ such that the restriction of f to the set $B = \{1, 2\}$ is a monotonic function $f_B : B \rightarrow B$, moreover, $f(0) \in B$. In addition to these functions, the only other functions $A \rightarrow A$ in T are the identity function and the constant function with value 0. For $m > 1$, the functions $A^m \rightarrow A$ in T are those of the form $g \cdot \text{pr}_i^{A^m}$, where $g : A \rightarrow A$ in T and $i \in \mathbf{m}$. The dagger operation is defined as follows. For all functions $f : A \rightarrow A$ in T other than $\mathbf{1}_A$ or the constant function z with value 0, $f^\dagger$ is the least fixed point of the function f_B . We define $\mathbf{1}_A^\dagger = z^\dagger = 0$. For functions $f = g \cdot \text{pr}_1^{A^{1+p}}$ with $g : A \rightarrow A$ in T and $p > 0$, we define $f^\dagger = g^\dagger \cdot \text{!}_{A^p}$, and for functions $f = g \cdot \text{pr}_{1+i}^{A^{1+p}}$, we let $f^\dagger = g \cdot \text{pr}_i^{A^p}$. (If f can be written in both ways, the two definitions give the same result.) It is now clear that the scalar parameter and fixed point identities hold. For functions $f : A^{n+p} \rightarrow A^n$ in T , where $n \neq 1$, we define the dagger operation so that the scalar pairing identity hold. Suppose now that $f, g : A \rightarrow A$ in T . If f or g is $\mathbf{1}_A$, or a constant function, then

$$(f \cdot g)^\dagger = f \cdot (g \cdot f)^\dagger \quad (10)$$

either holds trivially or follows from the scalar fixed point identity. If neither f nor g is $\mathbf{1}_A$ or the constant function z , then $(f \cdot g)^\dagger = (f_B \cdot g_B)^\dagger$ and $f \cdot (g \cdot f)^\dagger = f_B \cdot (g_B \cdot f_B)^\dagger$, where the dagger operation appearing on the right sides of the equations is the least fixed point operation on monotonic functions on B . Since the least fixed point operation satisfies the composition identity, we have established the *biscalar composition identity* (10) for all functions $f, g : A \rightarrow A$ in T . The fact that the *biscalar power identities* $(f^n)^\dagger = f^\dagger$, $n \geq 2$ hold for all $f : A \rightarrow A$ in T can be established in a similar way. Thus, since T is generated by functions $A \rightarrow A$, it follows from a result proved in [4] that T is an iteration theory, in fact a pointwise iteration theory. We still need to show that no cpo structure on A makes all functions in T monotonic. Indeed, if the least element of the cpo is 0, then let $h : A \rightarrow A$ in T such that $h(0)$ is maximal. Then

$h(x) = h(0)$ for all x , contradicting the fact that T contains a function which is the identity on B . If 0 is not least, then let h be a function in T with $h(0) = \perp$, the bottom. Then $h(\perp) = \perp$. But on B , we are free to choose any constant in B . We have thus proved

Theorem 5. *There exists a pointwise iteration theory T on the set $\mathbf{3}$ such that there is no cpo structure on $\mathbf{3}$ making each function in T monotonic.*

There also exists a pointwise iteration theory T on the set $A = \mathbf{3}$ which consists of monotonic functions with respect to the usual total order, but such that the dagger operation is not one of the two extremal fixed point operations. Let T be the subtheory of $\mathbf{F}_A$ generated by the constants $A^0 \rightarrow A$ and the functions $f, g : A \rightarrow A$, $f(0) = f(1) = 1$, $f(2) = 2$, $g(0) = 0$, $g(1) = g(2) = 1$. Then $f \cdot g = g \cdot f$ is the constant function $u : A \rightarrow A$ with value 1, and $f^2 = f$, $g^2 = g$. Thus, together with the identity function and the three constant functions $A \rightarrow A$, T has six functions $A \rightarrow A$. For $m > 0$, any function $A^m \rightarrow A$ in T can be written as $h \cdot \text{pr}_i^{A^m}$, for some $i \in \mathbf{m}$ and $h : A \rightarrow A$ in T . Define the scalar dagger operation on T in the following way: $\mathbf{1}_A^\dagger = 1$, $f^\dagger = 2$, $g^\dagger = 0$. On the constant functions $A \rightarrow A$, the definition of the dagger operation is forced by the fixed point identity, so that, e.g., $u^\dagger = 1$. For functions $A^{1+p} \rightarrow A$ with $p > 0$, define the scalar dagger operation so that the scalar parameter identity hold: $(h \cdot \text{pr}_1^{A^{1+p}})^\dagger = h^\dagger \cdot \mathbf{1}_{A^p}$, $(h \cdot \text{pr}_{1+i}^{A^{1+p}})^\dagger = h \cdot \text{pr}_i^{A^p}$. Finally, for functions $A^{n+p} \rightarrow A^n$, where $n \neq 1$, define the dagger operation so that the scalar pairing identity hold. It is easy to check that T satisfies the biscalar composition identity. Since $h^2 = h$, for all $h : A \rightarrow A$ in T , also the biscalar power identities hold. Moreover, since T contains all of the constant functions $A^0 \rightarrow A$, the dagger operation is pointwise. Thus we have:

Theorem 6. *There is a pointwise iteration theory T on set $\mathbf{3}$ such that all functions in T are monotonic with respect to the usual total order on $\mathbf{3}$, but such that the dagger operation is not one of the two extremal fixed point operations.*

5 An Iteration Theory of Boolean Functions with a Non-pointwise Dagger

In this section we show that the theory T_0 generated by the binary function of conjunction $\wedge$ and the constant 0 can be turned into iteration theory with a non-pointwise dagger. The functions $f : \mathbf{2}^m \rightarrow \mathbf{2}$ in T_0 have the following form:

$$f(x_1, \dots, x_m) = \bigwedge_{i \in I} x_i, \quad (11)$$

for all $x_1, \dots, x_m \in \mathbf{2}$, where the set $I \subseteq \mathbf{m}$ only depends on f . We define the empty conjunction to be 0 (and not 1), so that when $I = \emptyset$, (11) defines $f(x_1, \dots, x_m) = 0$.

Suppose now that $f : \mathbf{2}^{1+p} \rightarrow \mathbf{2}$ in T_0 , say $f(x_1, \dots, x_{1+p}) = \wedge_{i \in I} x_i$, for all $x_1, \dots, x_{1+p} \in \mathbf{2}$. We define $f^\dagger(x_1, \dots, x_p) = \wedge_{1+i \in I} x_{1+i}$. When $f =$

$\langle f_1, \dots, f_n \rangle : \mathbf{2}^{n+p} \rightarrow \mathbf{2}^n$ in T_0 with $n \neq 1$, $f^\dagger$ is defined so that the scalar pairing identity hold. The dagger operation on T_0 is not pointwise. Indeed, let $f : \mathbf{2}^2 \rightarrow \mathbf{2}$ be the conjunction function, i.e., $f(x, y) = x \wedge y$, for all $x, y \in \mathbf{2}$. Then $f^\dagger(y) = y$, for all $y \in \mathbf{2}$, so that $f^\dagger = \mathbf{1}_2$. But $f(x, 1) = x$, for all $x \in \mathbf{2}$, and $f^\dagger(1) = 1 \neq 0 = \mathbf{1}_2^\dagger$. The fact that T_0 is an iteration theory can be shown by proving that T_0 is a quotient of the iteration theory of trees [2] over the signature containing only one binary symbol.

Theorem 7. *There exists an iteration theory of boolean functions with a non-pointwise dagger.*

References

1. H. Bekić, Definable operations in general algebras, and the theory of automata and flowcharts, *Technical Report*, IBM Laboratory, Vienna, 1969.
2. S. L. Bloom and Z. Ésik, *Iteration Theories: The Equational Logic of Iterative Processes*, Springer, 1993.
3. S. L. Bloom and Z. Ésik, The equational logic of fixed points, *Theoretical Computer Science*, 179(1997), 1–60.
4. S. L. Bloom and Z. Ésik, There is no finite axiomatization of iteration theories, in: proc. *LATIN 2000*, LNCS 1776, Springer, 2000, 367–376.
5. J. W. de Bakker and D. Scott, A theory of programs. IBM, Vienna, 1969.
6. Z. Ésik, Group axioms for iteration, *Information and Computation*, 148(1999), 131–180.
7. D. Niwinski, Equational μ -calculus, *Computation Theory, Zaborów, 1984*, LNCS 208, Springer, 1985, 169–176.
8. E. L. Post, *The Two-Valued Iterative Systems of Mathematical Logic*, Princeton University Press, 1941.

Appendix

In this Appendix we define the various derived operations that have been used in the sequel. Suppose that T is a theory. A *base morphism* in T is any morphism belonging to the least subtheory of T . It is easy to see that the base morphisms are the tuplings of the projections, i.e., the morphisms of the form $\langle \text{pr}_{i_1}^n, \dots, \text{pr}_{i_m}^n \rangle : n \rightarrow m$. In any theory T , the base morphisms $\text{pr}_1^{n,m} = \langle \text{pr}_1^{n+m}, \dots, \text{pr}_n^{n+m} \rangle : n+m \rightarrow n$, $\text{pr}_2^{n,m} = \langle \text{pr}_{n+1}^{n+m}, \dots, \text{pr}_{n+m}^{n+m} \rangle : n+m \rightarrow m$ determine a product diagram. Note that $\text{pr}_1^{n,0} = \text{pr}_2^{0,n} = \mathbf{1}_n$, for all $n \geq 0$. Given any $f : p \rightarrow n$ and $g : p \rightarrow m$, the unique mediating morphism $p \rightarrow n+m$ is given by $\langle \text{pr}_1^n \cdot f, \dots, \text{pr}_n^n \cdot f, \text{pr}_1^m \cdot g, \dots, \text{pr}_m^m \cdot g \rangle$. We denote this morphism by $\langle f, g \rangle$. The *pairing operation* thus defined is easily seen to be associative with neutral morphisms $!_n$. Thus, expressions like $\langle f, g, h \rangle$, where f, g and h have the same source, make sense. We also define $\text{pr}_i^{n,m,p} = \text{pr}_i^{n,m} \cdot \text{pr}_1^{n+m,p}$, $i = 1, 2$, $\text{pr}_3^{n,m,p} = \text{pr}_2^{m,p} \cdot \text{pr}_2^{n,m+p}$. Another associative operation is the $\times$ operation defined by $f \times g = \langle f \cdot \text{pr}_1^{p,q}, g \cdot \text{pr}_2^{p,q} \rangle$, for all $f : p \rightarrow n$ and $g : q \rightarrow m$.

An Algorithm Constructing the Semilinear Post* for 2-Dim Reset/Transfer VASS (Extended Abstract)

A. Finkel and G. Sutre

LSV, ENS Cachan & CNRS UMR 8643, France.
{finkel, sutre}@lsv.ens-cachan.fr

Abstract. The main result of this paper is a proof that the reachability set (post*) of any 2-dim Reset/Transfer Vector Addition System with States is an effectively computable semilinear set. Our result implies that numerous boundedness and reachability properties are decidable for this class. Since the traditional Karp and Miller's algorithm does not terminate when applied to 2-dim Reset/Transfer VASS, we introduce, as a tool for the proof, a new technique to construct a finite coverability tree for this class.

1 Introduction

Context. Model checking consists in verifying that a model — usually a transition system — satisfies a property (usually a temporal logic formula). It has become very popular because it is fully automatic when the transition system has a finite state space. However, many programs cannot be modelled by a finite-state transition system, for instance communication protocols for which the size of channels is not known in advance, distributed systems parametrized by the number of processes. The success of model checking for finite-state transition systems has suggested that an efficient verification technology could be developed for infinite-state transition systems as well.

Model checking infinite-state transition systems often reduces to effective computation of two potentially infinite sets: the set of predecessors pre^* and the set of successors post^* . In order to effectively compute post^* (resp. pre^*), one generally needs to find a class $\mathcal{C}$ of (finitely describable) infinite sets which has the following good properties: (1) closure under union, (2) closure under post (resp. pre) and (3) membership and inclusion are decidable with an elementary complexity.

We focus in this paper on programs with integer variables, or more precisely on several kinds of counter automata. Basically, counters hold nonnegative integer values and they can be modified through the following operations: increment (+1), decrement (-1), zero-test, reset and transfer (from one counter to another). Hence the infinite state space is $Q \times \mathbb{N}^k$, where Q is the set of control states. It is well known that 2 counters automata with increment, decrement and zero-test are sufficient to simulate any Turing machine. Thus, post^* and pre^* may be nonrecursive for this class.

Upward closed sets enjoy the three good required properties for the class of lossy counter automata and hence they are particularly well suited to the computation of post^* and pre^* for this class [BM99]. However, upward closed sets do not allow to exactly represent pre^* and post^* for general (nonlossy) counter automata. The class of semilinear sets (which contains upward closed sets) enjoy the three good required properties for general counter automata. However, even for 3 counters automata with only increment and decrement, post^* and pre^* may be nonsemilinear. Hence, our objective is to find classes of 2-counters automata for which post^* and pre^* are effectively computable semilinear sets.

Related Work. Vector Addition Systems with States (VASS) (or equivalently Petri nets) correspond to the class of counter automata with increment and decrement. For the class of VASS, post^* and pre^* are recursive [May84,Kos82]. Hopcroft and Pansiot proved that for 2-dim VASS (i.e. VASS with 2 counters), post^* — and hence pre^* — are semilinear and constructible¹. However, for 3-dim VASS, post^* and pre^* may be nonsemilinear [HP79].

Some papers recently appeared dealing with verification of n -dim Reset/Transfer VASS (VASS extended with resets and transfers), 2-dim VASS extended with zero-tests on one counter and Reset VASS with lossy counters:

counters	operations	post^*	pre^*
3	+1, -1 and resets	nonrecursive [DFS98]	nonrecursive [DFS98]
3 lossy	+1, -1 and resets	recursive, semilinear, non constructible [BM99]	semilinear, constructible [BM99]
2	+1, -1 and zero-tests on one counter	semilinear, constructible [FS00a]	semilinear, constructible [FS00a]
2	+1, -1, resets and transfers	recursive, semilinear [FS00a], constructible [this paper]	semilinear, constructible [FS00a]

We already know from [FS00a] that pre^* is semilinear and constructible for 2-dim Reset/Transfer VASS, and hence post^* is recursive ; we also know that post^* is semilinear for this class. However, this does not imply that post^* is constructible since for 3-dim Lossy Reset VASS, pre^* is a constructible semilinear set whereas post^* is a *non-constructible* semilinear set.

Our Contribution. Our main result is a technically nontrivial proof that post^* is semilinear and constructible for 2-dim Reset/Transfer VASS. The proof is achieved with the following four main steps.

1. We first prove that it is possible to compute the finite minimal coverability set of a 2-dim Reset VASS by using an extended Karp and Miller's algorithm and a recent technical result of Jančar [DJS99]. Notice that the traditional Karp and Miller's algorithm does not terminate for 2-dim Reset VASS [DFS98].

¹ We say that a semilinear set L is *constructible* when a Presburger formula for L is computable.

2. Then we obtain, by simulation, the minimal coverability set of a 2-dim Reset/Transfer VASS.
3. We show that any 2-dim Reset/Transfer VASS may be simulated by a 2-dim Extended VASS in the class $\mathbf{T}_1\mathbf{R}_2$ (i.e. VASS with the capacity of zero-testing on the first counter and reset on the second counter) preserving post^* (and also the same minimal coverability set).
4. We deduce from [FS00a] that we can build an algorithm for constructing the semilinear post^* of a 2-dim Reset/Transfer VASS.

We can deduce from this result that, for instance, reachability, reachability equivalence, boundedness and counter-boundedness are all decidable for the class of 2-dim Reset/Transfer VASS.

The proofs are technically non trivial and are omitted for space reasons. A complete presentation of these results can be found in the full paper [FS00b].

2 Reset/Transfer Vector Addition Systems with States

We write $[i..j]$ for the set $\{k \in \mathbb{N} / i \leq k \leq j\}$. If $x \in X^2$ and $\gamma \in \{1, 2\}$, we denote by $x(\gamma)$ the γ^{th} component of x . If $\gamma \in \{1, 2\}$, we write $\bar{\gamma}$ for the unique element of $\{1, 2\} \setminus \{\gamma\}$.

Let $\mathbb{N}_\omega$ denote the classical completion $\mathbb{N} \cup \{\omega\}$ of the set $\mathbb{N}$, where $n < \omega$ for all $n \in \mathbb{N}$. Operations are extended on $\mathbb{N}_\omega$ as follows: for every $z \in \mathbb{Z}$, we define $z + \omega = \omega + z = \omega$; and for every $x \in \mathbb{N}_\omega \setminus \{0\}$, we define $x \cdot \omega = \omega \cdot x = \omega$ and $0 \cdot \omega = \omega \cdot 0 = 0$. For every infinite nondecreasing sequence $(a_i)_{i \in \mathbb{N}}$ in $\mathbb{N}$, we define

$$\text{lub}(a_i) = \begin{cases} \omega & \text{if } \{a_i / i \in \mathbb{N}\} \text{ is infinite} \\ \text{Max}(\{a_i / i \in \mathbb{N}\}) & \text{otherwise} \end{cases}$$

Operations $+$, $\cdot$ and lub on $\mathbb{N}_\omega^2$ are componentwise extensions of the respective operations on $\mathbb{N}_\omega$ and we also extend $\leq$ on $\mathbb{N}_\omega^2$ (and $\mathbb{Z}^2$) as expected, with $v \leq v'$ if for all $\gamma \in \{1, 2\}$ we have $v(\gamma) \leq v'(\gamma)$. These two relations (written $\leq$) on $\mathbb{N}_\omega^2$ and $\mathbb{Z}^2$ are orderings. We write $z < z'$ when $z \leq z'$ and $z \neq z'$.

A *labelled transition system* is a structure $\mathcal{LTS} = (S, A, \rightarrow)$ where S is a set of *states*, A is a finite set of *actions* and $\rightarrow \subseteq S \times A \times S$ is a set of *transitions*. When S is finite, we say that $\mathcal{LTS}$ is a *finite* labelled transition system. A (finite) *path* π in a labelled transition system is any sequence of transitions $\pi = (s_0 \xrightarrow{\alpha_0} s'_0), (s_1 \xrightarrow{\alpha_1} s'_1), \dots, (s_k \xrightarrow{\alpha_k} s'_k)$ such that $s'_i = s_{i+1}$ for every $i \in [0..k-1]$ and π is shortly written $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \cdots s_k \xrightarrow{\alpha_k} s'_k$; moreover, we say that π is a *path from* s_0 *to* s'_k . We write $s \xrightarrow{*} s'$ when there exists a path from s to s' . For every subset $R \subseteq S$, we write $\text{post}^*(\mathcal{LTS}, R)$ (shortly for $\text{post}^*(R)$) for the set $\{s \in S / \exists r \in R, r \xrightarrow{*} s\}$ of *successors* of R .

We present 2-dim Reset/Transfer Vector Addition Systems with States in two steps : we first describe the *finite control structure* (noted **A**) of a 2-dim Reset/Transfer VASS and we then define a 2-dim Reset/Transfer VASS (noted

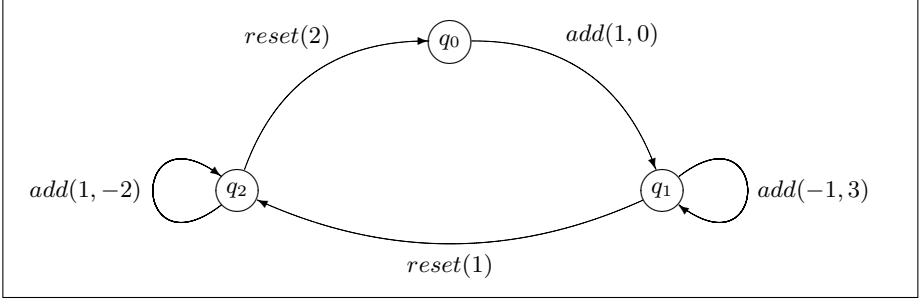


Fig. 1. A 2-dim Reset VASS

$\mathcal{A}$) as an *operational semantics* associated with such a control structure, which leads to a (potentially) infinite labelled transition system.

A *2-dim Reset/Transfer VASS Control* is a finite labelled transition system $A = (Q, Op, \rightarrow_A)$ such that $Op \subseteq \{add(z) \mid z \in \mathbb{Z}^2\} \cup \{reset(1), reset(2)\} \cup \{transfer(1 \rightarrow 2), transfer(2 \rightarrow 1)\}$.

Definition 2.1. A 2-dim Reset/Transfer VASS is a labelled transition system $\mathcal{A} = (S, Op, \rightarrow_{\mathcal{A}})$ based on a 2-dim Reset/Transfer VASS Control $A = (Q, Op, \rightarrow_A)$, where $S = Q \times \mathbb{N}^2$ is the set of states, Op is the set of actions, and the set of transitions $\rightarrow_{\mathcal{A}}$ is the smallest subset of $S \times Op \times S$ verifying the three following conditions:

1. if $q \xrightarrow{add(z)}_A q'$ then for all $a \in \mathbb{N}^2$ such that $a + z \geq 0$, $(q, a) \xrightarrow{add(z)}_{\mathcal{A}} (q', a + z)$
2. if $q \xrightarrow{reset(\gamma)}_A q'$ then for all $a \in \mathbb{N}^2$, $(q, a) \xrightarrow{reset(\gamma)}_{\mathcal{A}} (q', a')$ where $a'(\gamma) = 0$ and $a'(\bar{\gamma}) = a(\bar{\gamma})$
3. if $q \xrightarrow{transfer(\gamma \rightarrow \bar{\gamma})}_A q'$ then for all $a \in \mathbb{N}^2$, $(q, a) \xrightarrow{transfer(\gamma \rightarrow \bar{\gamma})}_{\mathcal{A}} (q', a')$ where $a'(\gamma) = 0$ and $a'(\bar{\gamma}) = a(\gamma) + a(\bar{\gamma})$.

Let $\mathcal{A}$ be a 2-dim Reset/Transfer VASS. States of $\mathcal{A}$ are called *control states* of $\mathcal{A}$ and transitions of $\mathcal{A}$ are called *control transitions* of $\mathcal{A}$. Every path $\pi = (q_0, a_0) \xrightarrow{\alpha_0} (q_1, a_1) \xrightarrow{\alpha_1} (q_2, a_2) \cdots (q_k, a_k) \xrightarrow{\alpha_k} (q_{k+1}, a_{k+1})$ in $\mathcal{A}$ is also shortly written $\pi = (q_0, a_0) \xrightarrow{\sigma} (q_{k+1}, a_{k+1})$ where $\sigma = q_0 \xrightarrow{\alpha_0} q_1 \xrightarrow{\alpha_1} q_2 \cdots q_k \xrightarrow{\alpha_k} q_{k+1}$ (notice that σ is a path in A).

Definition 2.2. A 2-dim Reset VASS (*resp.* 2-dim VASS) is any 2-dim Reset/Transfer VASS $\mathcal{A}$ whose control A contains no transfer transition (*resp.* no transfer transition and no reset transition).

3 Computation of the Minimal Coverability Set

We show in this section that the minimal coverability set of a 2-dim Reset/Transfer VASS is computable.

Let $\mathcal{A}$ be a 2-dim Reset/Transfer VASS with a set Q of control states and let (q_0, a_0) be an initial state of $\mathcal{A}$. We extend $\leq$ on $Q \times \mathbb{N}_\omega^2$ (and it is still an ordering) by $(q, v) \leq (q', v')$ if $q = q'$ and $v \leq v'$.

Definition 3.1. A coverability set of $(\mathcal{A}, (q_0, a_0))$ is any set $CS \subseteq Q \times \mathbb{N}_\omega^2$ satisfying the two following conditions:

1. for every $(q, a) \in \text{post}^*(q_0, a_0)$, there exists $(q', v') \in CS$ such that $(q, a) \leq (q', v')$
2. for every $(q, v) \in CS$, there exists an infinite nondecreasing sequence $(q, b_i)_{i \in \mathbb{N}}$ in $\text{post}^*(q_0, a_0)$ such that $v = \text{lub}(b_i)$

A coverability set CS is minimal if no proper subset of CS is a coverability set.

Notice that a minimal coverability set does not contain two comparable elements. The following theorem is an obvious consequence of the fact that 2-dim Reset/Transfer VASS are Well Structured Transition Systems [Fin90].

Theorem 3.2 ([Fin90]). Any 2-dim Reset/Transfer VASS $\mathcal{A}$ with an initial state (q_0, a_0) has a unique minimal coverability set, noted $\text{MCS}(\mathcal{A}, (q_0, a_0))$ (shortly $\text{MCS}(q_0, a_0)$), which is finite and satisfies $\text{MCS}(q_0, a_0) = \text{Max}(CS)$ for any finite coverability set CS .

3.1 Parametrized States and Path Schemes

We now focus on 2-dim Reset VASS and we present in this section parametrized states and path schemes. This whole section is highly inspired from [DJS99]. We have slightly adapted several definitions to our context.

For every $n \in \mathbb{N}$ and for every states $(q, a), (q', a')$ of a 2-dim Reset VASS $\mathcal{A}$, we write $(q, a) \xrightarrow{*}_n (q', a')$ when there exists a path from (q, a) to (q', a') containing at most n reset transitions. We also write $(q, a) \xrightarrow{* \text{reset}(\gamma)}_n (q', a')$ when there exists a path ending with a $\text{reset}(\gamma)$ from (q, a) to (q', a') and containing at most n reset transitions.

A parametrized state $s = \langle q, \gamma \rangle$ is a control state q equipped with a position $\gamma \in \{1, 2\}$. For every $x \in \mathbb{N}$, we write $s(x)$ for the state (q, γ_x) of $\mathcal{A}$, where $\gamma_x = (x, 0)$ if $\gamma = 1$ and $\gamma_x = (0, x)$ if $\gamma = 2$.

Definition 3.3. A path scheme of order n (shortly a path scheme) related to a parametrized state $s = \langle q, \gamma \rangle$ is a 3-tuple (σ, f, x_0) , where $\sigma : \mathbb{N} \rightarrow (\rightarrow_\Delta)^*$, $f : \mathbb{N} \rightarrow \mathbb{N}$ and $x_0 \in \mathbb{N}$, satisfying for every $x \geq x_0$ the three following conditions:

1. we have $s(x) \xrightarrow{\sigma(x)} s(f(x))$, and,
2. $\sigma(x)$ contains at most n reset control transitions, and,
3. $\sigma(x)$ ends with a reset control transition $p \xrightarrow{\text{reset}(\gamma)}_\Delta q$.

A path scheme (σ, f, x_0) is maximal if for every $x \geq x_0$ there is no $y > f(x)$ such that $s(x) \xrightarrow{* \text{reset}(\gamma)}_n s(y)$.

Observe that the *definition domains* of f and σ , $\text{Dom}(f)$ and $\text{Dom}(\sigma)$ respectively, necessarily contain the set $\{x \in \mathbb{N} \mid x \geq x_0\}$.

Definition 3.4. A parametrized state $s = \langle q, \gamma \rangle$ is n -sensible if there exists $x_0 \in \mathbb{N}$ such that $\{y \in \mathbb{N} \mid s(x) \xrightarrow{* \text{reset}(\gamma)}_n s(y)\}$ is finite for every $x \geq x_0$.

Notice that a parametrized state s is n -sensible if and only if there exists a maximal path scheme of order n related to s . We will use in the following the notion of *regular* path schemes, first introduced in [DJS99].

Definition 3.5 ([DJS99]). A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is d -regular, where $d \in \mathbb{N}$, if there exists $\rho \in \mathbb{Q}^+$ and a function $h : [0..d-1] \rightarrow \mathbb{Q}$ such that for every $x \in \text{Dom}(f)$, $f(x) = \rho x + h(x \bmod d)$.

Definition 3.6 ([DJS99]). A path scheme (σ, f, x_0) is regular if f is d -regular for some d , and for each $i \in [0..d-1]$ there exist $m \in \mathbb{N}$, $u_1, v_1, u_2, v_2, \dots, u_m, v_m, u_{m+1} \in (\rightarrow_\Lambda)^*$ and m d -regular functions $g_1, g_2, \dots, g_m$ such that for every $x \geq x_0$, if $x \bmod d = i$ then we have $\sigma(x) = u_1 v_1^{g_1(x)} u_2 v_2^{g_2(x)} \dots u_m v_m^{g_m(x)} u_{m+1}$.

According to [DJS99], the set $RPS(s)$ of regular path schemes related to a parametrized state s is recursive. Hence, for every parametrized state s , one may enumerate $RPS(s)$. We can easily derive from [DJS99] that:

Lemma 3.7. Let s be a parametrized state and let $n \in \mathbb{N}$. There exists a maximal regular path scheme of order n related to s if and only if s is n -sensible.

3.2 Coverability Tree for 2-Dim Reset VASS

Definition 3.8. An unbounded parametrized state is a pair (s, x_0) where s is a parametrized state, $x_0 \in \mathbb{N}$ and such that there exists a regular path scheme (σ, f, x_0) related to s satisfying $\forall x \geq x_0, f(x) > x$.

Note that for every d -regular function f , given by $f(x) = \rho x + h(x \bmod d)$, the property $\forall x \geq x_0, f(x) > x$ is decidable.

We start by showing that the minimal coverability set of a 2-dim Reset VASS is computable. In order to compute a coverability set for a 2-dim Reset VASS, we run *several* algorithms *in parallel*, as given by the `MinimalCoverabilitySet` algorithm. First, using the `RegularPathSchemes` algorithm, we enumerate regular path schemes in order to find unbounded parametrized states. At the same time, we develop a coverability tree using Karp and Miller's strategy and using also the list of (currently discovered) unbounded parametrized states (see the `CoverabilityTree` algorithm).

The `CoverabilityTree` algorithm constructs a tree in a breadth first manner (see line 3 of the algorithm). Lines 9–12 of the algorithm come from the following observation:

- if (s, x_0) is an unbounded parametrized state then we have $\text{lub}((f^i(x))_{i \in \mathbb{N}}) = \omega$, and
- moreover, for every $x \geq x_0$, if $s(x) \in \text{post}^*(q_0, a_0)$ then for every $i \in \mathbb{N}$, $s(f^i(x)) \in \text{post}^*(q_0, a_0)$.

Algorithm 1 RegularPathSchemes($\mathcal{A}, s, \text{var } L$)

Input: a 2-dim Reset VASS $\mathcal{A}$, a parametrized state s and a list L of unbounded parametrized states

- 1: **for each** regular path scheme (σ, f, x_0) related to s **do**
- 2: **if** $\forall x \geq x_0, f(x) > x$ **then**
- 3: add (s, x_0) to L
- 4: **return**

Algorithm 2 CoverabilityTree($\mathcal{A}, (q_0, a_0), \text{var } L$)

Input: a 2-dim Reset VASS $\mathcal{A}$, an initial state (q_0, a_0) and a list L of unbounded parametrized states

- 1: create root labelled (q_0, a_0)
- 2: **while** there are unmarked nodes **do**
- 3: pick an unmarked node $t : (q, v)$ among those of smallest depth
- 4: **if** there is a marked node $t' : (q, v')$ such that $v' \geq v$ **then**
- 5: skip {goto 18}
- 6: **else**
- 7: **if** there is an ancestor $t' : (q, v')$ of t such that $v \geq v'$, and $\omega > v(\gamma) > v'(\gamma)$ for some $\gamma \in \{1, 2\}$, and there is no reset between t' and t **then**
- 8: $v \leftarrow v + \omega \cdot (v - v')$
- 9: **if** $v \in \mathbb{N} \times \{0\}$ and there exists $x_0 \leq v(1)$ such that $((q, (\cdot, 0)), x_0) \in L$ **then**
- 10: $v \leftarrow (\omega, 0)$
- 11: **if** $v \in \{0\} \times \mathbb{N}$ and there exists $x_0 \leq v(2)$ such that $((q, (0, \cdot)), x_0) \in L$ **then**
- 12: $v \leftarrow (0, \omega)$
- 13: {We now create sons for the immediate successors of (q, v) }
- 14: **for each** transition $q \xrightarrow{\text{add}(z)} q'$ in $\mathbf{A}$ such that $v + z \geq 0$ **do**
- 15: construct a son $t' : (q', v + z)$ of t and label the arc $t \xrightarrow{\text{add}(z)} t'$
- 16: **for each** transition $q \xrightarrow{\text{reset}(\gamma)} q'$ in $\mathbf{A}$ **do**
- 17: construct a son $t' : (q', v')$ of t , where $v'(\gamma) = 0$ and $v'(\bar{\gamma}) = v(\bar{\gamma})$ and label the arc $t \xrightarrow{\text{reset}(\gamma)} t'$
- 18: mark t

Algorithm 3 MinimalCoverabilitySet($\mathcal{A}, (q_0, a_0)$)

Input: a 2-dim Reset VASS $\mathcal{A}$ and an initial state (q_0, a_0)

- 1: $L \leftarrow \emptyset$
- 2: **for each** parametrized state s **do**
- 3: {We start a new process enumerating regular path schemes related to s }
- 4: **start** RegularPathSchemes($\mathcal{A}, s, \text{var } L$)
- 5: {We run in parallel the CoverabilityTree algorithm}
- 6: **start** CoverabilityTree($\mathcal{A}, (q_0, a_0), \text{var } L$)
- 7: **wait until** CoverabilityTree terminates
- 8: **stop** all the processes RegularPathSchemes started at line 4
- 9: $CS \leftarrow \{(q, v) / t : (q, v) \text{ is a node in the tree constructed by CoverabilityTree}\}$
- 10: **return** Max(CS)

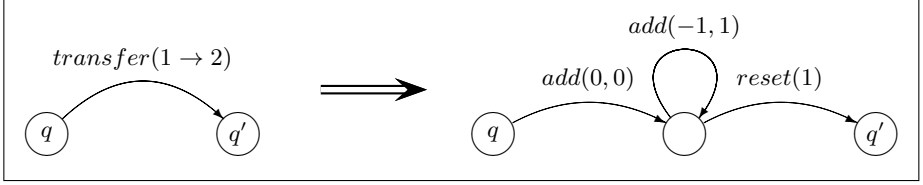


Fig. 2. Weak simulation of $transfer(1 \rightarrow 2)$ transitions by $reset(1)$ and add transitions

Intuitively, in order to show that the developed coverability tree is finite, we show that if there was an infinite branch in the tree, then an unbounded parametrized state should have been discovered (at some time) and, hence, the `CoverabilityTree` algorithm should have used this unbounded parametrized state to finish the branch, a contradiction. Thus, we get:

Proposition 3.9. *For any 2-dim Reset VASS $\mathcal{A}$ with an initial state (q_0, a_0) , the `MinimalCoverabilitySet` algorithm applied to $(\mathcal{A}, (q_0, a_0))$ terminates and we have:*

$$\text{MinimalCoverabilitySet}(\mathcal{A}, (q_0, a_0)) = \text{MCS}(q_0, a_0)$$

Now, given a 2-dim Reset/Transfer VASS $\mathcal{A}$, we construct a 2-dim Reset VASS $\mathcal{A}'$ obtained from $\mathcal{A}$ using the “weak simulation” of transfer transitions by reset transitions (and add transitions) given on Figure 2 (the “weak simulation” of $transfer(2 \rightarrow 1)$ transitions by $reset(2)$ and add transitions is obtained by symmetry). Observe that this simulation preserves the minimal coverability set (but not the set of successors). Hence, we obtain:

Theorem 3.10. *For any 2-dim Reset/Transfer VASS $\mathcal{A}$ with an initial state (q_0, a_0) , the minimal coverability set $\text{MCS}(q_0, a_0)$ is computable.*

Example 3.11. The minimal coverability set of the 2-dim Reset VASS of Figure 1 with initial state $(q_0, (0, 0))$ is $\{(q_0, (\omega, 0)), (q_1, (\omega, \omega)), (q_2, (\omega, \omega))\}$.

4 Computation of Post^*

We now focus on the computation of post^* for 2-dim Reset/Transfer VASS. We already proved in [FS00a] that for any 2-dim Reset/Transfer VASS $\mathcal{A}$ with a semilinear set S_0 of initial states, $\text{post}^*(S_0)$ is semilinear. However, the effective computability of post^* for this class was left an open problem in [FS00a] and we show in this section that post^* is constructible.

We first define semilinear sets. For any finite subset $P = \{p_1, p_2, \dots, p_k\}$ of $\mathbb{N}^2$ and for any $b \in \mathbb{N}^2$ we define $(b + P^*) = \{b + \sum_{i=1}^k x_i p_i \mid \forall i \in [1, k], x_i \in \mathbb{N}\}$ and $(b + P^*)$ is called a *linear* set. A *semilinear* set is a finite union of linear sets. We will say that a semilinear set L is *constructible* if there exists an algorithm

which computes a finite family $\{(b_i, P_i) / i \in I\}$ such that $L = \bigcup_{i \in I} (b_i + P_i^*)$. If $B \subseteq \mathbb{N}^2$, we shortly write $(B + P^*)$ for $\bigcup_{b \in B} (b + P^*)$.

We will also need the following definitions taken from [FS00a]. The class $\mathbf{T}_1\mathbf{R}_2$ of 2-dim *Extended VASS* is the set of 2-dim VASS extended with $test(1)$ and $reset(2)$ transitions. The operational semantics associated with a $test(1)$ transition, which actually represents a zero-test on counter 1, is as follows:

if $q \xrightarrow{test(1)}_{\mathcal{A}} q'$ then for all $a \in \mathbb{N}^2$, we have $(q, a) \xrightarrow{test(1)}_{\mathcal{A}} (q', a)$ iff $a(1) = 0$

Let $\mathcal{A}$ be a 2-dim Extended VASS in the class $\mathbf{T}_1\mathbf{R}_2$ with an initial state (q_0, a_0) . We define the set $\Omega(\mathcal{A}, (q_0, a_0))$ as the set of control states q such that $\{x \in \mathbb{N} / \exists q', (q_0, a_0) \xrightarrow{*} (q', (0, x)) \xrightarrow{test(1)} (q, (0, x))\}$ is infinite. Recall that:

Theorem 4.1 ([FS00a]). *For any 2-dim Extended VASS $\mathcal{A}$ in the class $\mathbf{T}_1\mathbf{R}_2$ with an initial state (q_0, a_0) , $post^*(q_0, a_0)$ is semilinear. Moreover, if $\Omega(\mathcal{A}, (q_0, a_0))$ is effectively computable, then $post^*(q_0, a_0)$ is constructible.*

Any 2-dim Reset/Transfer VASS $\mathcal{A}$ can be simulated by a 2-dim Extended VASS $\mathcal{A}'$ in the class $\mathbf{T}_1\mathbf{R}_2$, and $\Omega(\mathcal{A}', (q_0, a_0))$ may be computed from $MCS(\mathcal{A}, (q_0, a_0))$. Thus, we get:

Theorem 4.2. *For any 2-dim Reset/Transfer VASS $\mathcal{A}$ with a semilinear set S_0 of initial states, $post^*(S_0)$ is semilinear and it is constructible.*

Example 4.3. For the 2-dim Reset VASS of Figure 1, the set of successors of $(q_0, (0, 0))$ is:

$$\begin{aligned} post^*(q_0, (0, 0)) = & \quad q_0 \times ((0, 0) + \{(1, 0)\}^*) \\ & \cup q_1 \times (\{(1, 0), (0, 3)\} + \{(1, 0), (0, 3)\}^*) \\ & \cup q_2 \times (\{(0, 0), (1, 1), (2, 2)\} + \{(3, 0), (0, 3)\}^*) \end{aligned}$$

5 Conclusion

We have given an algorithm which computes the semilinear set $post^*$ of any 2-dim Reset/Transfer VASS. This result completes those obtained for pre^* in [FS00a]. Recall that for 3-dim Lossy Reset VASS, pre^* is a constructible semilinear set whereas $post^*$ is a *non-constructible* semilinear set (even if it is recursive). In other words, since $post^*$ was already known to be semilinear for 2-dim Reset/Transfer VASS [FS00a], we have proved that the following language is recursive, whereas it is not for 3-dim Lossy Reset VASS:

$$\{(A, L) / \text{post}^*(\mathcal{A}, s_0) = L \text{ where } \mathcal{A} \text{ is a 2-dim Reset/Transfer VASS} \\ \text{with initial state } (q_0, a_0) \text{ and } L \text{ is a semilinear set}\}$$

Our result is tight since it cannot be generalized to the extended class of 2-dim Reset/Transfer VASS where one adds zero-tests on one counter. We know from [FS00a] that $post^*$ is not even recursive for this extended class.

This result implies that numerous boundedness and reachability properties are decidable for this class. For instance, the computation of post^* allows to decide the following problems (which cannot be solved by only using pre^*): boundedness, counter-boundedness, containment of a semilinear set in post^* , inclusion and equivalence between two post^* sets.

Because 2-dim Reset/Transfer VASS are closed under intersection with finite automata, we obtain that LTL on finite sequences is decidable for this class. Since pre^* is also constructible [FS00a], we get that the existential fragment $\{\text{EF}, \text{EX}\}$ of CTL is decidable for 2-dim Reset/Transfer VASS.

Let us remark that line 2 in Algorithm 3 enumerates all regular path schemes related to a parameterized state s . This lead to a non-elementary complexity in the **MinimalCoverabilitySet** algorithm. We don't know whether it is possible to compute a minimal coverability set with an elementary complexity.

Our algorithm may also allow us to develop a strategy to over-approximate the reachability set of any two counters automaton $\mathcal{A}$ which is able to increase, decrease, zero-test, reset and transfer the 2 counters (recall the reachability set $\text{post}^*(\mathcal{A})$ is not in general recursive). This strategy, which allows to test safety properties, can be generalized to n -dim *Extended VASS* and to the pre^* operator.

References

- [BM99] A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. 16th Ann. Symp. Theoretical Aspects of Computer Science (STACS'99)*, Trier, Germany, Mar. 1999, volume 1563 of *Lecture Notes in Computer Science*, pages 323–333. Springer, 1999.
- [DFS98] C. Dufourd, A. Finkel, and Ph. Schnoebelen. Reset nets between decidability and undecidability. In *Proc. 25th Int. Coll. Automata, Languages, and Programming (ICALP'98)*, Aalborg, Denmark, July 1998, volume 1443 of *Lecture Notes in Computer Science*, pages 103–115. Springer, 1998.
- [DJS99] C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of Reset P/T nets. In *Proc. 26th Int. Coll. Automata, Languages, and Programming (ICALP'99)*, Prague, Czech Republic, July 1999, volume 1644 of *Lecture Notes in Computer Science*, pages 301–310. Springer, 1999.
- [Fin90] A. Finkel. Reduction and covering of infinite reachability trees. *Information and Computation*, 89(2):144–179, 1990.
- [FS00a] A. Finkel and G. Sutre. Decidability of reachability problems for classes of two counters automata. In *Proc. 17th Ann. Symp. Theoretical Aspects of Computer Science (STACS'2000)*, Lille, France, Feb. 2000, volume 1770 of *Lecture Notes in Computer Science*, pages 346–357. Springer, 2000.
- [FS00b] A. Finkel and G. Sutre. Verification of two counters automata. Research Report LSV-2000-7, Lab. Specification and Verification, ENS de Cachan, Cachan, France, June 2000.
- [HP79] J. Hopcroft and J. J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979.
- [Kos82] S. R. Kosaraju. Decidability of reachability in vector addition systems. In *Proc. 14th ACM Symp. Theory of Computing (STOC'82)*, San Francisco, CA, May 1982, pages 267–281, 1982.
- [May84] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

NP-Completeness Results and Efficient Approximations for Radiocoloring in Planar Graphs

D.A. Fotakis, S.E. Nikolettseas, V.G. Papadopoulou, and P.G. Spirakis *

Computer Technology Institute (CTI) and Patras University, Greece
Riga Fereou 61, 26221 Patras, Greece

Abstract. The Frequency Assignment Problem (FAP) in radio networks is the problem of assigning frequencies to transmitters exploiting frequency reuse while keeping signal interference to acceptable levels. The FAP is usually modelled by variations of the graph coloring problem. The Radiocoloring (RC) of a graph $G(V, E)$ is an assignment function $\Phi : V \rightarrow \mathbb{N}$ such that $|\Phi(u) - \Phi(v)| \geq 2$, when u, v are neighbors in G , and $|\Phi(u) - \Phi(v)| \geq 1$ when the minimum distance of u, v in G is two. The discrete number and the range of frequencies used are called order and span, respectively. The optimization versions of the Radiocoloring Problem (RCP) are to minimize the span or the order. In this paper we prove that *the min span RCP is NP-complete for planar graphs*. Next, we provide an $O(n\Delta)$ time algorithm ($|V| = n$) which obtains a radiocoloring of a planar graph G that *approximates the minimum order within a ratio which tends to 2* (where Δ the maximum degree of G). Finally, we provide a *fully polynomial randomized approximation scheme* (fpras) for the *number of valid radiocolorings of a planar graph G with λ colors*, in the case $\lambda \geq 4\Delta + 50$.

1 Introduction

The Problem of Frequency Assignment in radio networks (FAP) is a well-studied, interesting problem. It is usually modelled by variations of graph coloring. The interference between transmitters are usually modelled by the interference graph $G(V, E)$, where the set V corresponds to the set of transmitters and E represents distance constraints. The set of colors represents the available frequencies. In addition, the color of each vertex in a particular assignment gets an integer value which has to satisfy certain inequalities compared to the values of colors of nearby nodes in the interference graph G (frequency-distance constraints). We here study a variation of FAP, called the Radiocoloring Problem (RCP), that models co-channel and adjacent interference constraints.

* This research is partially supported by the European Union Fifth Framework Programme Projects ALCOM-FT, ARACNE and the Greek GSRT PENED'99 Project ALKAD.

Definition 1. (Radiocoloring) Given a graph $G(V, E)$ consider a function $\Phi : V \rightarrow N^*$ such that $|\Phi(u) - \Phi(v)| \geq 2$ if $D(u, v) = 1$ and $|\Phi(u) - \Phi(v)| \geq 1$ if $D(u, v) = 2$. The least possible number of colors (order) that can be used to radio-color G is denoted by $X_{\text{order}}(G)$. The number $\nu = \max_{v \in V} \Phi(v) - \min_{u \in V} \Phi(u) + 1$ is called span of the radiocoloring of G and the least such number is denoted as $X_{\text{span}}(G)$.

In most real life cases the network topology formed has some special properties, e.g. G is a lattice network or a planar graph. We remark that although there are some papers on the problem of Radiocoloring for general graphs [5,7], there are only few works for Radiocoloring of graphs with some special characteristics such as the planar graphs.

Real networks usually reserve bandwidth (range of frequencies) rather than distinct frequencies. The objective of an assignment here is to minimize the bandwidth (span) used. The optimization version of RCP related to this objective, called *min span RCP*, tries to find a radiocoloring for G of minimum span, $X_{\text{span}}(G)$. However, there are cases where the objective is to minimize the distinct number of frequencies used so that unused frequencies are available for other use by the system. The related optimization version of RCP here, called *min order RCP*, tries to find a radiocoloring that uses the minimum number of distinct frequencies, $X_{\text{order}}(G)$. The *min span order RCP* tries to find one from all minimum span assignments that uses the minimum number of colors. The min order span RCP is defined similarly, by interchanging span to order.

Another variation of FAP considers the square of a given a graph $G(V, E)$, G^2 . This is the graph of the same vertex set V and an edge set $E' : \{u, v\} \in E'$ iff $D(u, v) \leq 2$ in G . The problem is to color the square, G^2 of a graph G with the minimum number of colors, denoted as $\chi(G^2)$. This problem is studied in [12] named as *Distance-2-Coloring* (D2C). Observe that for any graph G , the minimum order of the min order RCP of G , $X_{\text{order}}(G)$, is the same as the (vertex) chromatic number of G^2 , i.e. $X_{\text{order}}(G) = \chi(G^2)$. However, notice that, the set of colors used in the computed assignments of the two problems are different. The colors of the distance one vertices in the RCP should be at frequency distance two instead of one in the coloring of the G^2 . However, from a valid coloring of G^2 we can always reach a valid RC of G by doubling the assigned color of each vertex. Observe also that $\chi(G^2) \leq X_{\text{span}} \leq 2\chi(G^2)$.

In [12] it is proved that the distance-2-coloring (D2C) for planar graphs is NP-complete. We remark that D2C problem is different from min span order RCP. To see this, recall that in RCP the span and order are different metrics since the distance one and two constraints are different in RCP. Note also, that the objectives of D2C and min span order RCP are different (the order and the span, respectively). Therefore, the minimum order of D2C for a given G may be different (smaller) than the minimum order of the min span order RCP of G . In [6], both by combinatorial arguments and exhaustive search, it is proved that the two problems are different. As an example, see the instance of figure 1. The minimum order of D2C for this graph is 6, while the minimum order of min span order RCP of G is 8. Note finally, that since D2C is equivalent to min order

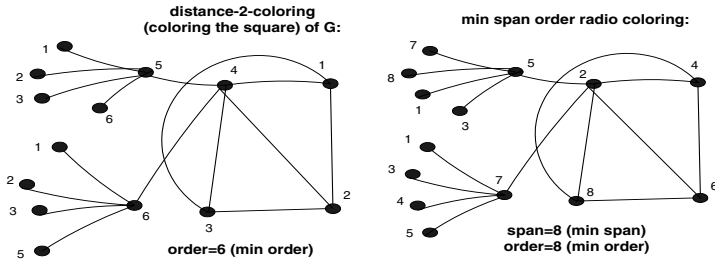


Fig. 1. An instance of G where the minimum order of distance-2-coloring and min span order RCP are different (equal to 6 and 8 respectively)

span RCP as far as the order is concerned, the min order RCP also computes a different order from the order computed in min span order RCP.

Thus, the NP-completeness of distance-2-coloring certainly does not imply the NP-completeness of min span order RCP which is here proved to be NP-complete. Additionally, the NP-completeness proof of [12] does not work for planar graphs of maximum degree $\Delta > 7$. Hence, its proof gives no information on the complexity of distance-2-coloring of planar graphs of maximum degree > 7 . In contrast, our NP-completeness proof works for planar graphs of all degrees. In [12,11] a 9-approximation algorithm for the distance-2-coloring of planar graphs is presented.

In [7] it has been proved that the problem of radiocoloring is NP-complete, even for graphs of diameter 2. The reductions use highly non-planar graphs.

In [3] a similar problem for *planar* graphs has been considered. This is the *Hidden Terminal Interference Avoidance (HTIA)* problem, which requests to color the vertices of a planar graph G so that vertices at *minimum distance exactly 2* get different colors. In [3] this problem is shown to be NP-complete.

However, the above mentioned result does not imply the NP-hardness of the min span order RCP which is proved here to be NP-complete. This so because HTIA is a different problem from RCP; in HTIA it is allowed to color neighbors in G with the same color while in RCP the colors of neighbor vertices should be at frequency distance at least two apart. Thus, the minimum number of colors as well as span needed for HTIA can vary arbitrarily from the $X_{\text{order}}(G)$ and $X_{\text{span}}(G)$. To see this consider e.g. the t -size clique graph K_t . In HTIA this can be colored with only one color. In our case (RCP) we need t colors and span of size $2t$ for K_t . In addition, the reduction used by [3], heavily exploits the fact that neighbors in G *get the same color* in the component substitution part of the reduction. Consequently, their reduction considers a different problem and it cannot be easily modified to produce an NP-hardness proof of RCP.

In this paper we are interested in *min span RCP* and in *min order RCP* of a *planar* graph G .

(a) We show that *the problem of min span RCP is NP-complete* for planar graphs. As we argued before, this result is *not* implied by the NP-completeness results of similar problems (distance-2-coloring or HTIA) [3,12].

(b) We then present an $O(n\Delta)$ algorithm that *approximates* the minimum order of RCP, X_{order} of a planar graph G by a constant ratio which tends to 2 as the maximum degree of G increases. Our algorithm is motivated by a constructive coloring theorem presented by Heuvel and McGuinness ([8]). Their construction can easily lead (as we show) to an $O(n^2)$ technique assuming that a planar embedding of G is given. We improve the time complexity of the approximation, and we present a much more simple algorithm to verify and implement. Our algorithm does not need any planar embedding as input.

(c) We also study the problem of *estimating the number of different radiocolorings* of a planar graph G . This is a #P-complete problem (as can be easily seen from our completeness reduction that can be done parsimonious). We use here standard techniques of rapidly mixing Markov Chains and the *newer method of coupling* for proving *rapid convergence* (see e.g. [9]) and we present a *fully polynomial randomised approximation scheme* for estimating the number of radiocolorings with λ colors for a planar graph G , when $\lambda \geq 4\Delta + 50$.

2 The NP-Completeness of the RCP for Planar Graphs

In this section, we show that the decision version of min span radiocoloring remains NP-complete for planar graphs. The decision version of min span radio coloring is, given planar graph G and an integer B , to decide whether there exists a valid radiocoloring for G of span no more than B . Therefore, the optimization version of min span radiocoloring, that is to compute a valid radiocoloring of minimum span, remains NP-hard for planar graphs.

Theorem 1. *The min span radiocoloring problem is NP-complete for planar graphs.*

Proof. The decision version of min span radio coloring clearly belongs to the class NP. To prove the theorem, we transform PLANAR-3-COLORING to min span radiocoloring. The PLANAR-3-COLORING problem is, given a planar graph $G(V, E)$, to determine whether the vertices of G can be colored with three colors, such that no adjacent vertices get the same color.

We consider a plane embedding of $G(V, E)$. Let $F(G)$ be the set of faces of G , and Δ_G be the maximum degree of G . Also, for a face $f \in F(G)$, let $size(f)$ be the number of edges of f . We define an integer Γ as $\Gamma = \max\{\Delta_G, \max_{f \in F(G)}\{size(f)\}\}$. Then, given a plane embedding of $G(V, E)$, we construct in polynomial time another planar graph $G'(V', E')$, such that there exists a radiocoloring for G' of span no more than $\Gamma + 5$ iff G is 3-colorable.

The graph $G'(V', E')$ has four kinds of vertices and three kinds of edges. As for the vertices, V' is the union of the following sets:

1. The vertex set V of the original graph G . These vertices are also called *existing* vertices, and the corresponding set is denoted by $V_E = V$.
2. The set of *intermediate* vertices V_I . There exists one intermediate vertex i_{uv} for each edge (u, v) of the original graph G .

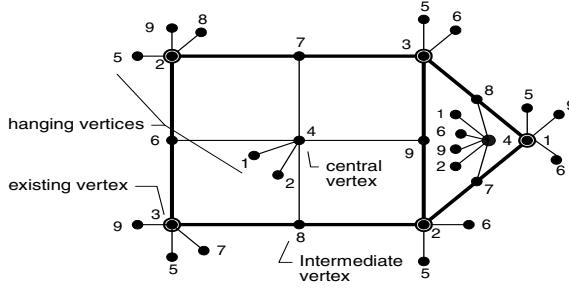


Fig. 2. The graph G' obtained from an instance G of the planar 3-coloring problem ($\Gamma = 4$)

3. The set of *central* vertices V_C . There exists one central vertex w_f for each face $f \in F(G)$.
4. The set of *hanging* vertices V_H . The hanging vertices are used to increase the degree of the existing and the central vertices to $\Gamma + 1$ and $\Gamma + 2$ respectively. For each $w_e \in V_E$ (resp. $w_c \in V_C$), an appropriate set of vertices $V_H(w_e)$ (resp. $V_H(w_c)$) is added so that $\deg_{G'}(w_e) = \Gamma + 1$ (resp. $\deg_{G'}(w_c) = \Gamma + 2$).

Additionally, G' contains the following sets of edges:

- The set E_I replacing the edge set E of the original graph G . In particular, we replace each $(u, v) \in E$ with two edges (u, i_{uv}) and (i_{uv}, v) connecting the corresponding intermediate vertex to the endpoints u, v of the original edge.
- The set E_C consisting of the edges connecting the central vertices with the intermediate ones. In particular, each $i_{uv} \in V_I$ is connected with exactly one of the central vertices corresponding to the faces that (u, v) belongs to.
- The set E_H consisting of the edges connecting the existing and central vertices to the corresponding hanging ones. In particular, we connect each $w \in V_E$ (resp. $w \in V_C$) with all the hanging vertices $V_H(w)$. Then, $\deg_{G'}(w)$ must be $\Gamma + 1$ (resp. $\Gamma + 2$).

The construction of G' is completed by removing any central vertices (and the corresponding hanging ones) not connected to any intermediate vertices. Fig. 2 demonstrates a simple example of the construction above. The edges of graph G are marked with heavy lines and its vertices with bigger cycles.

By carefully studying the distance constraints in the resulting graph, we reach a radio coloring with no more than $\Gamma + 5$ colors (see [6]). Thus, we have:

Claim. If the graph G is 3-colorable, then the suggested radiocoloring for the resulting graph G' has span no more than $\Gamma + 5$. (See [6] for a proof.) $\square$

Claim. If there exists a radiocoloring of G' of span $\Gamma + 5$ then G is 3-colorable. (See [6] for a proof.) $\square$

(end of proof of Theorem 1) $\square$

Corollary 1. The min span order radiocoloring problem is NP-complete for planar graphs. $\square$

3 A Constant Ratio Approximation Algorithm

We provide here an approximation algorithm for radiocoloring of planar graphs by modifying the constructive proof of the theorem presented by Heuvel and McGuinness in [8]. Our algorithm is easier to verify with respect to correctness than what the proof given by [8] suggests. It also has better time complexity (i.e. $O(n\Delta)$) compared to the (implicit) algorithm in [8] which needs $O(n^2)$ and also assumes that a planar embedding of the graph is given. The improvement was achieved by performing the heavy part of the computation of the algorithm only in some instances of G instead of all as in [8]. This enables less checking and computations in the algorithm. Also, the behavior of our algorithm is very simple and more time efficient for graphs of small maximum degree. Finally, the algorithm provided here needs no planar embedding of G , as opposed to the algorithm implied in [8].

Very recently and independently, Agnarsson and Halldórsson [2] presented approximations for the chromatic number of square and power graphs (G^k). Their method does not explicitly present an algorithm. A straightforward implementation is difficult and not efficient. Also, the performance ratio for planar graphs of general Δ obtained is also 2.

The theorem of [8] states that a planar graph G can be radio-colored with at most $2\Delta + 25$ colors. More specifically, [8] considers the problem of $L_{-}(p,q)$ -Labeling, which is defined as following:

Definition 2. $L_{-}(p,q)$ -Labeling Find an assignment $\phi : V \rightarrow \{0, 1, \dots, \nu\}$, called $L_{-}(p,q)$ -Labeling, which satisfies: $|\phi(u) - \phi(v)| \geq p$ if $\text{dist}_G(u, v) = 1$ and $|\phi(u) - \phi(v)| \geq q$ if $\text{dist}_G(u, v) = 2$.

Definition 3. The minimum ν for which an $L_{-}(p,q)$ -Labeling exists is denoted by $\lambda(G; p, q)$ (the p, q -span of G).

The main theorem of [8] is the following:

Theorem 2. ([8]) If G is a planar graph with maximum degree Δ and p, q are positive integers with $p \geq q$, then $\lambda(G; p, q) \leq (4q - 2)\Delta + 10p + 38q - 23$.

By setting $p = q = 1$ and using the observation $\lambda(G; 1, 1) = \chi(G^2)$, where $\chi(G^2)$ is the usual chromatic number of the graph G^2 (defined in the Introduction section), we get immediately, as also [8] notices, that:

Corollary 2. If G is a planar graph with maximum degree Δ then $\chi(G^2) \leq 2\Delta + 25$.

The theorem is proved using two Lemmata. The first of the two Lemmata is the following:

Lemma 1. ([8]) Let G be a simple planar graph. Then there exists a vertex v with k neighbors $v_1, v_2, \dots, v_k$ with $d(v_1) \leq \dots \leq d(v_k)$ such that one of the following is true:

- (i) $k \leq 2$; (ii) $k = 3$ with $d(v_1) \leq 11$; (iii) $k = 4$ with $d(v_1) \leq 7$ and $d(v_2) \leq 11$;
- (iv) $k = 5$ with $d(v_1) \leq 6$, $d(v_2) \leq 7$, and $d(v_3) \leq 11$.

The second Lemma, which is quite similar (see [8]). These two Lemmata give the so-called *unavoidable configurations* of G . The following operations apply to G : For an edge $e \in E$ let G/e denote the graph obtained from G by contracting e . For a vertex $v \in V$ let $G * v$ denote the graph obtained by deleting v and for each $u \in N(v)$ adding an edge between u and u^- and between u and u^+ (if these edges do not exist in G already). The notation $N(v)$ denotes the neighbors of v . The notation u^- , with $u^- \in N(v)$, denotes the edge vu^- which directly precedes edge vu (moving clockwise), and u^+ , with $u^+ \in N(v)$, refers to the edge vu^+ which directly succeeds edge vu .

The two Lemmata are used to define the graph H , a vertex $v \in V(G)$ and an edge $e \in E(G)$ using the rules explained in [8]. The main idea is to define H to be $H = G/e$ or $H = G * v$, with $e = vv_1$ and $d(v) \leq 5$, depending on which cases of the two Lemmata holds, so that always $\Delta(H) \leq \Delta$. Using these observations it is proved, by induction, that the minimum (p, q) -span needed for the $L_-(p, q)$ -Labeling of H is less or equal to $\lambda(G; p, q) \leq (4q - 2)\Delta + 10p + 38q - 23$.

From H we can easily return to G as follows. If $H = G/e$ then let v' the new vertex created from the contraction of edge e . In this case, in G we set $v_1 = v'$ (this is a valid assumption since $\text{degree}(v_1) \leq \text{degree}(v')$). Now we only need to color vertex v (for both cases of $H = G/e$ or $H = G * v$). From the way v was chosen, we have $d(v) \leq 5$, and it is easily seen that it can be colored with one of the $\lambda(G; p, q) \leq (4q - 2)\Delta + 10p + 38q - 23$ colors.

For the case of radiocoloring of a planar graph G , we can use $p=1$ and $q=1$ for the order, thus, the above theorem states that we need at most $2\Delta + 25$ colors.

3.1 Our Algorithm

We will use only lemma 1 and the operation G/e in order to provide a much more simple and more efficient algorithm than what implied in [8].

Radiocoloring(G)

[I] Sort the vertices of G by their degree.

[II] If $\Delta \leq 12$ then follow procedure (1) below:

Procedure (1): Every planar graph G has at least one vertex of degree ≤ 5 . Now, inductively assume that any proper (in vertices) subgraph of G can be radiocolored by 66 colors. Consider a vertex v in G with $\text{degree}(v) \leq 5$. Delete v from G to get G' . Now recursively radiocolor G' with 66 colors. The number of colors that v has to avoid is at most $5\Delta + 5 \leq 65$. Thus, there is one free color for v .

[III] If $\Delta > 12$ then

1. Find a vertex v and a neighbor v_1 of it, as described in Lemma 1, and set $e = vv_1$.
2. Form $G' = G/e$ ($G' = (V', E')$ with $|V'| = n - 1$, while $|V| = n$) and denote the new vertex in G' obtained by the contraction of edge e as v' .

Modify the sorted degrees of G by deleting v, v_1 , and inserting v' at the appropriate place, and also modify the possible affected degrees of the neighbors of both v and v_1 .

3. $\phi(G') = \text{Radiocoloring}(G')$
4. Extend $\phi(G')$ to a valid radiocoloring of G :
 - (a) Set $v_1 = v'$ and give to v_1 the color of v' .
 - (b) Color v with one of the colors used in the radiocoloring ϕ of G' .

3.2 Correctness, Time Efficiency and Approximation Ratio of the Algorithm

Notice first for procedure (1) that it implies a proper coloring of G^2 with $X = 66$ colors. Then, assign the frequencies $1, 3, \dots, 2X - 1$ to the obtained color classes of G^2 . This is a proper radiocoloring of G with the same number of colors.

Proposition 1. *The algorithm $\text{radiocoloring}(G)$ outputs a valid radiocoloring for G using no more than $\max\{66, 2\Delta + 25\}$ colors.*

Proof. By induction, see [6] for details. □

Lemma 2. *Our algorithm approximates $X_{\text{order}}(G)$ by a constant factor of at most $\max\{2 + \frac{25}{\Delta}, \frac{66}{\Delta}\}$.* □

Lemma 3. *Our algorithm runs in $O(n\Delta)$ sequential time.*
(See [6] for a proof.) □

4 An fpras for the Number of Radiocolorings of a Planar Graph

4.1 Sampling and Counting

Let G be a *planar* graph of maximum degree $\Delta = \Delta(G)$ on vertex set $V = \{0, 1, \dots, n - 1\}$ and C be a set of λ colors. Let $\phi : V \rightarrow C$ be a (proper) radiocoloring of the vertices of G . Such a radiocoloring always exists if $\lambda \geq 2\Delta + 25$ and can be found by our $O(n\Delta)$ time algorithm of the previous section.

Consider the Markov Chain (X_t) whose state space $\Omega = \Omega_\lambda(G)$ is the set of all radiocolorings of G with λ colors and whose transition probabilities from radiocoloring X_t are modelled by:

1. choose a vertex $v \in V$ and a color $c \in C$ uniformly at random (u.a.r.)
2. recolor vertex v with color c . If the resulting coloring X' is a *proper* radiocoloring then let $X_{t+1} = X'$ else $X_{t+1} = X_t$.

The procedure above is similar to the “Glauber Dynamics” of an antiferromagnetic Potts model at zero temperature, and was used by [9] to estimate the number of proper colorings of any low degree graph with k colors.

The Markov Chain (X_t) , which we refer to in the sequel as $M(G, \lambda)$, is *ergodic*, provided $\lambda \geq 2\Delta + 26$, in which case its stationary distribution is *uniform* over Ω . We show here that $M(G, \lambda)$ is *rapidly mixing* i.e. converges, in time polynomial in n , to a close approximation of the stationary distribution, provided that $\lambda \geq 2(2\Delta + 25)$. This can be used to get a fully polynomial randomised approximation scheme (fpras) for the number of radiocolorings of a planar graph G with λ colors, in the case where $\lambda \geq 4\Delta + 50$. For some definitions and measures used below see [6].

4.2 Rapid Mixing

As indicated by the (by now standard) techniques for showing rapid mixing by *coupling* ([9,10]), our strategy here is to construct a coupling for $M = M(G, \lambda)$, i.e. a stochastic process (X_t, Y_t) on $\Omega \times \Omega$ such that each of the processes (X_t) , (Y_t) , considered in isolation, is a faithful copy of M . We will arrange a joint probability space for (X_t) , (Y_t) so that, far from being independent, the two processes tend to *couple* so that $X_t = Y_t$ for t large enough. If coupling can occur rapidly (independently of the initial states X_0, Y_0), we can infer that M is rapidly mixing, because the variation distance of M from the stationary distribution is bounded above by the probability that (X_t) and (Y_t) *have not* coupled by time t (see e.g. [1,4]).

The transition $(X_t, Y_t) \rightarrow (X_{t+1}, Y_{t+1})$ in the coupling is defined by the following experiment:

1. Select $v \in V$ uniformly at random (u.a.r.).
2. Compute a permutation $g(G, X_t, Y_t)$ of C according to a procedure to be explained.
3. Choose a color $c \in C$ u.a.r.
4. In the radiocoloring X_t (respectively Y_t) recolor vertex v with color c (respectively $g(c)$) to get a new radiocoloring X' (respectively Y')
5. If X' (respectively Y') is a proper radiocoloring then $X_{t+1} = X'$ (respectively $Y_{t+1} = Y'$), else let $X_{t+1} = X_t$ (respectively $Y_{t+1} = Y_t$).

Note that, whatever procedure is used to select the permutation g , the distribution of $g(c)$ is *uniform*, thus (X_t) and (Y_t) are both faithful copies of M .

We now remark that any set of vertices $F \subseteq V$ can have the same color in the graph G^2 only if they can have the same color in some radiocoloring of G . Thus, given a proper coloring of G^2 with λ' colors, we can construct a proper radiocoloring of G by giving the values (new colors) $1, 3, \dots, 2\lambda' - 1$ in the color classes of G^2 . Note that this transformation preserves the number of colors (but not the span).

Now let $A = A_t \subseteq V$ be the set of vertices on which the colorings of G^2 implied by X_t, Y_t agree and $D = D_t \subseteq V$ be the set on which they disagree. Let $d'(v)$ be the number of edges incident at v in G^2 that have one point in A and one in D . Clearly, if m' is the number of edges of G^2 spanning A, D , we get $\sum_{v \in A} d'(v) = \sum_{v \in D} d'(v) = m'$.

The procedure to compute $g(G, X_t, Y_t)$ is as follows:

- (a) If $v \in D$ then g is the identity.
- (b) If $v \in A$ then proceed as follows: Denote by N the set of neighbors of v in G^2 . Define $C_x \subseteq C$ to be the set of all colors c , such that some vertex in N receives c in radiocoloring Y_t but no vertex in N receives c in radiocoloring X_t . Let C_y be defined as C_x with the roles of X_t, Y_t interchanged. Observe $C_x \cap C_y = \emptyset$ and $|C_x|, |C_y| \leq d'(v)$. Let, w.l.o.g., $|C_x| \leq |C_y|$. Choose any subset $C'_y \subseteq C_y$ with $|C'_y| \leq |C_x|$ and let $C_x = \{c_1, \dots, c_r\}, C'_y = \{c'_1, \dots, c'_r\}$ be enumerations of C_x, C'_y coming from the orderings of X_t, Y_t . Finally, let g be the permutation $(c_1, c'_1), \dots, (c_r, c'_r)$ which interchanges the color sets C_x, C'_y and leaves all other colors fixed.

It is clear that $|D_{t+1}| - |D_t| \in \{-1, 0, 1\}$. By careful and nontrivial estimations of the probability of $\Pr\{|D_{t+1}| = |D_t| \pm 1\}$ we finally get (see [6] for a full proof): $\Pr\{D_t \neq 0\} \leq n(1 - \alpha)^t \leq ne^{-\alpha t}$, where $\alpha = [\lambda - 2(2\Delta + 25)]/n > 0$ when $\lambda > 2(2\Delta + 25)$.

So, we note that $\Pr\{D_t \neq \emptyset\} \leq \epsilon$, where ϵ is upper bound on the variation distance at time t , provided that $t \geq \frac{1}{\alpha} \ln\left(\frac{n}{\epsilon}\right)$.

Theorem 3. *The above method leads to a fully polynomial randomized approximation scheme for the number of radiocolorings of a planar graph G with λ colors, provided that $\lambda > 2(2\Delta + 25)$, where Δ is the maximum degree of G . (See [6] for a proof.)* $\square$

5 Further Work

A major open problem is to get a polynomial time approximation to $X_{order}(G)$ of asymptotic ratio < 2 . The improvement of the time efficiency of the approximation procedure is also a subject of further work.

References

1. D. Aldous: Random walks in finite groups and rapidly mixing Markov Chains. *Seminaire de Probabilites XVII 1981/82* (A. Dold and B. Eckmann, eds), Springer Lecture Notes in Mathematics 986 (1982) 243-297.
2. Geir Agnarsson, Magnus M. Hallorsson: Coloring Powers of planar graphs. *Symposium of Discrete Algorithms* (2000).
3. Alan A. Bertossi and Maurizio A. Bonuccelli: Code assignment for hidden terminal interference avoidance in multihop packet radio networks. *IEEE/ACM Trans. Networking* 3, 4 (Aug. 1995) 441 - 449.
4. P. Diaconis: Group representations in probability and statistics. *Institute of Mathematical Statistics*, Hayward CA, (1988).
5. D. Fotakis, G. Pantziou, G. Pentaris and P. Spirakis: Frequency Assignment in Mobile and Radio Networks. *Networks in Distributed Computing, DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 45 American Mathematical Society (1999) 73-90.
6. D.A. Fotakis, S.E. Nikolettseas, V.G. Papadopoulou and P.G. Spirakis: NP-completeness Results and Efficient Approximations for Radiocoloring in Planar Graphs. *CTI Technical Report* (2000) ([url: http://www.cti.gr/RD1](http://www.cti.gr/RD1)).
7. D. Fotakis and P. Spirakis: Assignment of Reusable and Non-Reusable Frequencies. *International Conference on Combinatorial and Global Optimization* (1998).
8. J. Van D. Heuvel and S. McGuinness: Colouring the square of a Planar Graph. *CDAM Research Report Series*, Jule (1999).
9. M. Jerrum: A very simple algorithm for estimating the number of k -colourings of a low degree graph. *Random Structures and Algorithms* 7 (1994) 157-165.
10. M. Jerrum: Markov Chain Monte Carlo Method. *Probabilistic Methods for Algorithmic Discrete Mathematics*, Springer (1998).
11. S. Ramanathan, E. R. Loyd: Scheduling algorithms for Multi-hop Radio Networks. *IEEE/ACM Trans. on Networking*, 1(2): (April 1993) 166-172.
12. S. Ramanathan, E. R. Loyd: The complexity of distance2-coloring. *4th International Conference of Computing and information*, (1992) 71-74.

Explicit Fusions

Philippa Gardner and Lucian Wischik

Computing Laboratory, University of Cambridge,
Philippa.Gardner@cl.cam.ac.uk, ljw1004@cam.ac.uk

Abstract. We introduce *explicit fusions* of names. To ‘fuse’ two names is to declare that they may be used interchangeably. An explicit fusion is one that can exist in parallel with some other process, allowing us to ask for instance how a process might behave in a context where $x = y$. We present the π_F -calculus, a simple process calculus with explicit fusions. It is similar in many respects to the fusion calculus but has a simple local reaction relation. We give embeddings of the π -calculus and the fusion calculus. We provide a bisimulation congruence for the π_F -calculus and compare it with hyper-equivalence in the fusion calculus.

1 Introduction

We introduce *explicit fusions* of names. To ‘fuse’ two names is to declare that they may be used interchangeably. An *explicit* fusion is one that can exist in parallel with some other process. For example, we can use the explicit fusion $\langle x=y \rangle$ to ask how a process might behave in a context where the addresses x and y are equal.

In this paper we focus on one particular application of explicit fusions. We introduce the π_F -calculus, which incorporates these fusions. It is similar to the π -calculus in that it has input and output processes which react together. It differs from the π -calculus in how they react. In a π -reaction, names are sent by the output process to replace abstracted names in the input process; this replacement is represented with a substitution. In contrast a π_F -reaction is directionless and *fuses* names; this is recorded with an explicit fusion.

The π_F -calculus is similar in many respects to the fusion calculus of Parrow and Victor [10,13], and to the chi-calculus of Fu [1]. These calculi also have a directionless reaction which fuses names. The difference is in how the name-fusions have effect. In the fusion calculus, fusions occur implicitly within the reaction relation and their effect is immediate. In the π_F -calculus, fusions are explicitly recorded and their effect may be delayed. A consequence of this is that π_F -reaction is a simple local reaction between input and output processes.

Explicit fusions can be used to analyse, in smaller steps, reactions that occur in existing process calculi. We give embedding results for the π -calculus and the fusion calculus. These embeddings show that explicit fusions are expressive enough to describe both name-substitution in the π -reaction, and the fusions that occur in the fusion reaction. We are currently exploring an embedding of the λ -calculus in the π_F -calculus [14]. Intriguingly, explicit fusions allow for an embedding which is purely compositional, in contrast with the analogous embeddings in the π -calculus and fusion calculus.

We provide a bisimulation congruence for the π_F -calculus, which is automatically closed with respect to substitution. We compare it with hyper-equivalence in the fusion calculus [10] and open bisimulation in the π -calculus [12].

2 The π_F -Calculus

To illustrate the key features of the π_F -calculus, we contrast it to the fusion calculus. Both calculi have symmetric input and output processes. They have no abstraction operator. Instead, they interpret the π -calculus abstraction $\langle x \rangle P$ with the concretion $(\nu x) \langle x \rangle P$. A π_F -reaction is

$$z. \langle x \rangle P \mid \bar{z}. \langle y \rangle Q \mid R \searrow_{\pi_F} \langle x=y \rangle \mid P \mid Q \mid R.$$

The reaction in this example is a local one between the input and output processes. However the effect of the resulting fusion $\langle x=y \rangle$ is global in scope: x and y can be used interchangeably throughout the entire process, including R . To limit the scope of the fusion, we use restriction. For example, restricting x in the above expression we obtain

$$(\nu x) (\langle x=y \rangle \mid P \mid Q \mid R) \equiv P\{y/x\} \mid Q\{y/x\} \mid R\{y/x\}.$$

Thus, using just explicit fusions and restriction, we can derive a name-substitution operator which behaves like the standard capture-avoiding substitution.

The corresponding reaction in the fusion calculus *requires* that either x or y be restricted: for instance,

$$(\nu x) (z. \langle x \rangle P \mid \bar{z}. \langle y \rangle Q \mid R) \searrow_{fu} P\{y/x\} \mid Q\{y/x\} \mid R\{y/x\}.$$

The x and y are implicitly fused during the reaction. If we had restricted y rather than x , then the substitution would have been $\{x/y\}$. The full polyadic reaction rule, using many $\bar{x}$ s and $\bar{y}$ s, is more complicated.

We assume an infinite set of names ranged over by $u, \dots, z$, and write $\vec{z}$ for a sequence of names and $|\vec{z}|$ for its length.

Definition 1. The set $\mathcal{P}_{\pi_F}$ of processes of the π_F -calculus is defined by the grammar

$$P ::= \mathbf{nil} \mid P \mid P \mid (\nu x)P \mid \langle x \rangle \mid \langle x=y \rangle \mid x.P \mid \bar{x}.P$$

We call the process $\langle x \rangle$ a datum, and the process $\langle x=y \rangle$ a fusion.

We say that a datum is at the *top-level* if it is not contained within an input or output process. The *arity* of a process is the number of top-level datums in it. We write $P : m$ to declare that P has arity m . More general arities are also possible, such as typing information similar to the sorting discipline for the π -calculus [8]. For simplicity, we consider in this paper only that fragment of the π_F -calculus without replication or summation. Replication is considered elsewhere [14].

Datums are primitive processes, with the process $\langle \bar{y} \rangle \mid P$ corresponding to the conventional concretion $\langle \bar{y} \rangle P$. The choice between datums and concretions does not affect the results in this paper. Our choice to use datums is motivated in [2, 14], where we represent variables of the λ -calculus by datums to obtain a direct translation of the λ -calculus into the π_F -calculus.

The definitions of *free* and *bound* names are standard. The *restriction* operator $(\nu x)P$ binds x ; x is free in $\langle x \rangle$, $x.P$, $\bar{x}.P$ and in fusions involving x . We write $fn(P)$ to denote the set of free names in P . We use the following abbreviations: $(\nu \vec{x})P \stackrel{\text{def}}{=} (\nu x_1) \dots (\nu x_n)P$, $\langle \vec{x} \rangle \stackrel{\text{def}}{=} \langle x_1 \rangle \mid \dots \mid \langle x_n \rangle$ and $\langle \vec{x}=\vec{y} \rangle \stackrel{\text{def}}{=} \langle x_1=y_1 \rangle \mid \dots \mid \langle x_n=y_n \rangle$.

Standard axioms for $ $ and nil :		
$P \mathbf{nil} \equiv P$	$(P Q) R \equiv P (Q R)$	$P Q \equiv Q P$ if $P : 0$
Standard scope axioms:		
$(\nu x)(P Q) \equiv (\nu x)P Q$	if $x \notin fn(Q)$	$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
$(\nu x)(P Q) \equiv P (\nu x)Q$	if $x \notin fn(P)$	
Fusion axioms:		
$\langle x=x \rangle \equiv \mathbf{nil}$	$\langle x=y \rangle x.P \equiv \langle x=y \rangle y.P$	$\langle x=y \rangle \langle x \rangle \equiv \langle x=y \rangle \langle y \rangle$
$(\nu x)\langle x=y \rangle \equiv \mathbf{nil}$	$\langle x=y \rangle \bar{x}.P \equiv \langle x=y \rangle \bar{y}.P$	$\langle x=y \rangle z.P \equiv \langle x=y \rangle z.(\langle x=y \rangle P)$
$\langle x=y \rangle \equiv \langle y=x \rangle$	$\langle x=y \rangle \langle x=z \rangle \equiv \langle x=y \rangle \langle y=z \rangle$	$\langle x=y \rangle \bar{z}.P \equiv \langle x=y \rangle \bar{z}.(\langle x=y \rangle P)$

Fig. 1. The structural congruence between π_F -process, written $\equiv$, is the smallest equivalence relation satisfying these axioms and closed with respect to contexts

Definition 2. The structural congruence between processes, written $\equiv$, is the smallest congruence satisfying the axioms given in Figure 1, and closed with respect to the contexts $\_|\_, (\nu x)\_, x.\_$ and $\bar{x}.\_$

The side-condition on the commutativity of parallel composition allows for processes of arity 0 to be reordered, but not arbitrary processes. For instance,

$$x.P | \bar{x}.Q \equiv \bar{x}.Q | x.P \quad \text{but} \quad \langle x \rangle | \langle y \rangle | P \not\equiv \langle y \rangle | \langle x \rangle | P.$$

This is essentially the same as in the conventional π -calculus, where processes can be reordered but the names in the concretion $\langle xy \rangle P$ cannot.

The fusion axioms require further explanation. Our intuition is that $\langle x=y \rangle$ is an equivalence relation which declares that two names can be used interchangeably. The fusion $\langle x=x \rangle$ is congruent to the nil process. So too is $(\nu x)\langle x=y \rangle$, since the bound name x is unused. The final six fusion axioms describe small-step substitution, allowing us to deduce $\langle x=y \rangle | P \equiv \langle x=y \rangle | P\{y/x\}$ and α -conversion. For example,

$$\begin{aligned}
 & (\nu x)(\bar{x}.\mathbf{nil}) \\
 \equiv & (\nu x)(\nu y)(\langle x=y \rangle | \bar{x}.\mathbf{nil}) && \text{create fresh bound name } y \text{ as an alias for } x \\
 \equiv & (\nu x)(\nu y)(\langle x=y \rangle | \bar{y}.\mathbf{nil}) && \text{substitute } y \text{ for } x \\
 \equiv & (\nu y)(\bar{y}.\mathbf{nil}) && \text{remove the now-unused bound name } x
 \end{aligned}$$

Honda investigates a simple process framework with equalities on names that are probably the most like our fusion axioms [5]: the axioms are different but the spirit of the equalities is similar. Honda and Yoshida have also introduced π -processes called *equators* [6]. In the asynchronous π -calculus they simulate the effect of explicit fusions; but they do not generalise to the synchronous case [7].

With the structural congruence we can factor out the datums and fusions. In particular, every π_F -process is structurally congruent to one in the *standard form*

$$\langle \vec{u}=\vec{v} \rangle | (\nu \vec{x})(\langle \vec{y} \rangle | P),$$

where the $\vec{x}$ s are distinct and contained in the $\vec{y}$ s, and P contains no datums or fusions in its top level. We call $\langle \vec{u}=\vec{v} \rangle | (\nu \vec{x})(\langle \vec{y} \rangle | \_)$ the *interface* of the process. It is unique in the sense that, given two congruent standard forms

$$\langle \vec{u}_1=\vec{v}_1 \rangle | (\nu \vec{x}_1)(\langle \vec{y}_1 \rangle | P_1) \equiv \langle \vec{u}_2=\vec{v}_2 \rangle | (\nu \vec{x}_2)(\langle \vec{y}_2 \rangle | P_2),$$

the fusions $\langle \vec{u}_1=\vec{v}_1 \rangle$ and $\langle \vec{u}_2=\vec{v}_2 \rangle$ denote the same equivalence relation on names, $|\vec{x}_1| = |\vec{x}_2|$, and the datums $\vec{y}_1, \vec{y}_2$ are identical and the processes P_1, P_2 structurally congruent up to the name-equivalence and α -conversion of the $\vec{x}$ s. We write $E(P)$ for the name-equivalence. It can be inductively defined on the structure of processes, or more simply characterised by $(x, y) \in E(P)$ iff $P \equiv P|_{\langle x=y \rangle}$.

We define a symmetric *connection operator* $@$ between processes of the same arity, which connects them through their interfaces. The effect of the connection $P@Q$ is to fuse together the top-level names in P and Q . If P and Q have standard forms $\langle \vec{u}_1=\vec{v}_1 \rangle | (\nu \vec{x}_1)(\langle \vec{y}_1 \rangle | P_1)$ and $\langle \vec{u}_2=\vec{v}_2 \rangle | (\nu \vec{x}_2)(\langle \vec{y}_2 \rangle | P_2)$ respectively, then

$$P@Q \stackrel{\text{def}}{=} \langle \vec{u}_1 \vec{u}_2 = \vec{v}_1 \vec{v}_2 \rangle | (\nu \vec{x}_1 \vec{x}_2)(\langle \vec{y}_1 = \vec{y}_2 \rangle | P_1 | P_2),$$

renaming if necessary to avoid name clashes. Because interfaces are unique, the connection operator is well-defined up to structural congruence.

Definition 3. *The reaction relation between processes, written $\searrow$, is the smallest relation closed with respect to $|-|$, $(\nu x)_-$ and $_- \equiv \rightarrow$, which satisfies*

$$z.P \mid \bar{z}.Q \searrow P@Q.$$

3 Embedding the π -Calculus and the Fusion Calculus

The π_F -calculus naturally embeds the π -calculus, the π_I -calculus [11] and the fusion calculus. For the embeddings we consider the fragment of the calculus without summation or replication. The interesting part in the translations concerns the abstractions and concretions:

$$\begin{aligned} (\vec{x})P &\xrightarrow{*} (\nu \vec{x})(\langle \vec{x} \rangle | P^*) && \text{Abstraction} \\ (\nu \vec{x})(\langle \vec{z} \rangle | P) &\xrightarrow{*} (\nu \vec{x})(\langle \vec{z} \rangle | P^*) && \text{Concretion} \end{aligned}$$

For example, the π -reaction $z.(x)P \mid \bar{z}.(y)Q \searrow_{\pi} P\{y/x\} \mid Q$ corresponds to the π_F -reaction

$$\begin{aligned} z.(\nu x)(\langle x \rangle | P^*) \mid \bar{z}.(\langle y \rangle | Q^*) &\searrow_{\pi_F} (\nu x)(\langle x \rangle | P^*) @ (\langle y \rangle | Q^*) \\ &\equiv (\nu x)(\langle x=y \rangle | P^* | Q^*) && \text{renaming if necessary} \\ &\equiv (\nu x)(\langle x=y \rangle | P^*\{y/x\} | Q^*) && \text{substituting } y \text{ for } x \\ &\equiv P^*\{y/x\} \mid Q^* && \text{removing unused bound } x \end{aligned}$$

There is a key difference between the (straightforward) embeddings of the π - and π_I -calculi, and the embedding of the fusion calculus. For the π -calculus, reaction of a π_F -process in the image of $(\_)^*$ necessarily results in a process congruent to one in

the image. Even though the reaction temporarily results in a fusion $\langle x=y \rangle$, one of those fused names must have arisen from an abstraction $(x)Q$ and so the fusion can be factored away. The same is not true with the fusion calculus. For example,

$$z.(\langle x \rangle | P^*) \mid \bar{z}.(\langle y \rangle | Q^*) \searrow_{\pi_F} \langle x=y \rangle \mid P^* \mid Q^*.$$

The process on the left is in the image of the fusion calculus under $(-)^*$, but the one on the right has an unbounded explicit fusion and so is not. Essentially, because the fusion calculus has unbound input and output processes and yet lacks explicit fusions, it can only allow those reactions that satisfy certain restriction properties on names (given at the end of this section). We do obtain an embedding result in the sense that, by restricting x or y we obtain a π_F -reaction which corresponds to a valid fusion reaction. This embedding result is as strong as can be expected: the fusion reaction requires that a side-condition on restricted names be satisfied; the π_F -reaction does not.

Embedding the π -Calculus

We define a translation $(-)^*$ from π -processes to π_F -processes. We also define a reverse translation $(-)^o$ and prove embedding results. (The embedding of the π_I -calculus is similar.) Following [8], the set $\mathcal{P}_\pi$ of π -processes is generated by the grammar

$$\begin{array}{ll} P ::= \mathbf{nil} \mid P|P \mid (\nu x)P \mid z.A \mid \bar{z}.C & \text{Processes} \\ A ::= (\vec{x})P & \text{Abstractions} \\ C ::= (\nu \vec{x})(\vec{y})P & \text{Concretions} \end{array}$$

where the $\vec{x}$ s are distinct and, in the concretion, contained in the $\vec{y}$ s. The structural congruence on processes and the reaction relation are standard. In order to define the reverse translation $(-)^o$, we identify the π -image in $\mathcal{P}_{\pi_F}$:

$$\begin{array}{ll} P ::= \mathbf{nil} \mid P|P \mid (\nu x)P \mid A & \text{Processes} \\ A ::= z.(\nu \vec{x})(\langle \vec{x} \rangle | P) \mid \bar{z}.(\nu \vec{x})(\langle \vec{y} \rangle | P) & \text{Input / Output Processes} \end{array}$$

Definition 4. The translation $(-)^* : \mathcal{P}_\pi \rightarrow \mathcal{P}_{\pi_F}$ is defined inductively by

$$\begin{array}{ll} (\mathbf{nil})^* = \mathbf{nil} & (z.(\vec{x})P)^* = z.(\nu \vec{x})(\langle \vec{x} \rangle | P^*) \\ (P|Q)^* = P^*|Q^* & (\bar{z}.(\nu \vec{x})(\langle \vec{y} \rangle | P)^* = \bar{z}.(\nu \vec{x})(\langle \vec{y} \rangle | P^*) \\ ((\nu x)P)^* = (\nu x)P^* & \end{array}$$

The translation $(-)^o : \pi\text{-image} \rightarrow \mathcal{P}_\pi$ is the reverse of this.

Theorem 5. The translations $(-)^* : \mathcal{P}_\pi \rightarrow \mathcal{P}_{\pi_F}$ and $(-)^o : \pi\text{-image} \rightarrow \mathcal{P}_\pi$ are mutually inverse, preserve the structural congruence, and strongly preserve the reaction relation:

$$\begin{array}{l} P \in \mathcal{P}_\pi \text{ and } P \searrow_{\pi} Q \text{ implies } P^* \searrow_{\pi_F} Q^* \\ P \in \pi\text{-image and } P \searrow_{\pi_F} Q \text{ implies } P^o \searrow_{\pi} R \text{ and } R^* \equiv_{\pi_F} Q \end{array}$$

Embedding the Fusion Calculus

The set of fusion processes $\mathcal{P}_{fu}$ is generated by the grammar

$$P ::= \mathbf{nil} \mid P|P \mid (\nu x)P \mid z\vec{x}.P \mid \bar{z}\vec{x}.P.$$

Its structural congruence is standard, and its reaction relation is generated by the rule

$$(\nu \vec{u})(z\vec{x}.P \mid \bar{z}\vec{y}.Q \mid R) \searrow P\sigma \mid Q\sigma \mid R\sigma,$$

where $\text{ran}(\sigma), \text{dom}(\sigma) \subseteq \{\vec{x}, \vec{y}\}$ and $\vec{u} = \text{dom}(\sigma) \setminus \text{ran}(\sigma)$ and $\sigma(v) = \sigma(w)$ if and only if $(v, w) \in E(\langle \vec{x} = \vec{y} \rangle)$. The side-conditions describe a natural concept. Consider the equivalence relation generated from the equalities $\vec{x} = \vec{y}$. The side-conditions ensure that, for each equivalence class, every element is mapped by σ to a single free witness.

The *fusion-image* of the fusion calculus in the π_F -calculus is similar to that of the π -calculus, but with input and output processes given by

$$A ::= z.(\langle \vec{y} \rangle | P) \mid \bar{z}.(\langle \vec{y} \rangle | P) \quad \text{Input / Output Processes}$$

Translations between the fusion calculus and the fusion-image are straightforward.

Theorem 6. *The translations $(-)^* : \mathcal{P}_{fu} \rightarrow \mathcal{P}_{\pi_F}$ and $(-)^o : \mathcal{P}_{\pi_F} \rightarrow \mathcal{P}_{fu}$ are mutually inverse and preserve structural congruence as in Theorem 5. They also preserve reaction in the sense that*

$$\begin{aligned} P \in \mathcal{P}_{fu} \text{ and } P \searrow_{fu} Q \text{ implies } P^* \searrow_{\pi_F} Q^* \\ P \in \text{fusion-image and } P \searrow_{\pi_F} Q \text{ implies } \exists \vec{u}. (\nu \vec{u})P \searrow_{fu} R \text{ and } R^* \equiv_{\pi_F} (\nu \vec{u})Q \end{aligned}$$

As discussed, reaction of a process in the fusion-image does not necessarily result in a process also in the fusion-image. Note that the restricted names $\vec{u}$ are precisely those needed to satisfy the side-conditions on reaction in the fusion calculus.

4 Bisimulation for the π_F -Calculus

We define a bisimulation relation for the π_F -calculus using a labelled transition system (LTS) in the standard way. The LTS consists of the usual CCS labels $\bar{x}$, x and τ , accompanied by a definition of bisimulation which incorporates fusions:

$$PSQ : 0 \text{ implies for all } x, y, \text{ if } \langle x=y \rangle | P \xrightarrow{\alpha} P_1 \text{ then } \langle x=y \rangle | Q \xrightarrow{\alpha} Q_1 \text{ and } P_1 SQ_1.$$

We call this bisimulation the *open bisimulation*, by analogy with open bisimulation for the π -calculus.

In this definition of open bisimulation, labelled transitions are analysed with respect to all possible fusion contexts $- \mid \langle x=y \rangle$. In fact, we do not need to consider all such contexts. Instead we introduce *fusion transitions*, generated by the axiom

$$x.P \mid \bar{y}.Q \xrightarrow{?x=y} P @ Q.$$

$$\begin{array}{c}
 x.P \xrightarrow{x} P \quad \bar{x}.P \xrightarrow{\bar{x}} P \\
 \\
 x.P \mid \bar{y}.Q \xrightarrow{?x=y} P@Q \quad x.P \mid \bar{x}.Q \xrightarrow{\tau} P@Q \\
 \\
 \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \quad \frac{P \xrightarrow{\alpha} P'}{Q|P \xrightarrow{\alpha} Q|P'} \quad \frac{P \xrightarrow{\alpha} P', \quad x \notin \alpha}{(\nu x)P \xrightarrow{\alpha} (\nu x)Q} \quad \frac{P' \equiv P \xrightarrow{\alpha} Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}
 \end{array}$$

Fig. 2. *Quotiented* labelled transition system. We do not distinguish between $?x=y$ and $?y=x$. The final rule closes the LTS with respect to the structural congruence

The label $?x=y$ declares that the process can react in the presence of an explicit fusion $\langle x=y \rangle$. Fusion transitions allow us to define bisimulation without having to quantify over fusion contexts. However, the label also declares additional information about the structure of the process. If $P \xrightarrow{?x=y} Q$, then we infer that P must contain input and output processes on unbounded channels x and y . In order to define a bisimulation relation which equals the open bisimulation, we remove this additional information:

$$PSQ:0 \text{ and } P \xrightarrow{?x=y} \text{ implies either } Q \xrightarrow{?x=y} Q_1 \text{ or } Q \xrightarrow{\tau} Q_1, \text{ and } \langle x=y \rangle | P_1 \mathcal{S} \langle x=y \rangle | Q_1$$

The resulting bisimulation equals open bisimulation. A consequence of adding fusion transitions is that we can use standard techniques to prove congruence.

We give two labelled transition systems for the π_F -calculus: a *quotiented LTS* in which we explicitly close the labelled transitions with respect to the structural congruence, and a *structured LTS* in which the labelled transitions are defined according to the structure of processes. These LTSs are equivalent; the quotiented LTS is simpler to understand, and the structured LTS is easier to use. We define corresponding bisimulation relations and prove that they are the same. Finally we use the structured LTS to prove that bisimulation is a congruence.

The Quotiented LTS

The quotiented LTS is given in Figure 2. Notice that the structural congruence rule allows fusions to affect the labels on transitions: for example, the process $\langle x=y \rangle | \bar{x}.P$ can undergo the transition $\xrightarrow{\bar{y}}$ as well as $\xrightarrow{\bar{x}}$, because it is structurally congruent to $\langle x=y \rangle | \bar{y}.P$. We have defined transitions for arbitrary processes instead of just processes of arity 0. This requires two rules for parallel composition, since parallel composition does not commute in the presence of datums.

Proposition 7. $P \searrow Q \text{ iff } P \xrightarrow{\tau} Q$.

We now define the bisimulation relation. Our basic intuition is that two processes are bisimilar if and only if they have the same interface and, in all contexts of the form $\_@ \langle \bar{y} \rangle$, if one process can do a labelled transition then so can the other. In fact we do not need to consider all such contexts. Instead it is enough to factor out the top-level datums and analyse the labelled transitions for just the processes of arity 0.

$x.P \xrightarrow{x}_s P$	$\bar{x}.P \xrightarrow{\bar{x}}_s P$	$\frac{P \xrightarrow{\alpha_1}_s Q \quad \alpha_1 =_{E(P)} \alpha_2}{P \xrightarrow{\alpha_2}_s Q} *$
$\frac{P \xrightarrow{x}_s P' \quad Q \xrightarrow{\bar{y}}_s Q'}{P Q \xrightarrow{?x=y}_s P'@Q'}$	$\frac{P \xrightarrow{\bar{x}}_s P' \quad Q \xrightarrow{y}_s Q'}{P Q \xrightarrow{?x=y}_s P'@Q'}$	$\frac{P \xrightarrow{?x=x}_s Q}{P \xrightarrow{\tau}_s Q}$
$\frac{P \xrightarrow{\alpha}_s P'}{P Q \xrightarrow{\alpha}_s P' Q}$	$\frac{P \xrightarrow{\alpha}_s P'}{Q P \xrightarrow{\alpha}_s Q P'}$	$\frac{P \xrightarrow{\alpha}_s Q, \quad x \notin \alpha}{(\nu x)P \xrightarrow{\alpha}_s (\nu x)Q}$

* We write $\alpha =_{E(P)} \beta$ if α, β are identical up to $E(P)$

Fig. 3. *Structured* labelled transition system. This LTS does not include a rule involving the structural congruence. Recall that $E(P)$ is the equivalence relation on names generated by P . A simple characterisation is given by $(x, y) \in E(P)$ if and only if $P \equiv P|(x=y)$

Definition 8 (Fusion bisimulation). A symmetric relation $\mathcal{S}$ is a fusion bisimulation iff whenever PSQ then

1. $P, Q : m > 0$ implies P and Q have standard forms $\langle \vec{u}=\vec{v} \rangle | (\nu \vec{x}) (\langle \vec{y} \rangle | P_1)$ and $\langle \vec{u}=\vec{v} \rangle | (\nu \vec{x}) (\langle \vec{y} \rangle | Q_1)$ respectively and $\langle \vec{u}=\vec{v} \rangle | P_1 \mathcal{S} \langle \vec{u}=\vec{v} \rangle | Q_1$;
2. $P, Q : 0$ implies they have standard forms $\langle \vec{u}=\vec{v} \rangle | P_1$ and $\langle \vec{u}=\vec{v} \rangle | Q_1$, and
 - a) if $P \xrightarrow{\alpha} P'$ where α is $x, \bar{x}$ or τ , then $Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{S} Q'$
 - b) if $P \xrightarrow{?x=y} P'$ then either $Q \xrightarrow{?x=y} Q'$ or $Q \xrightarrow{\tau} Q'$, and $\langle x=y \rangle | P' \mathcal{S} \langle x=y \rangle | Q'$;
3. similarly for Q .

Two processes P and Q are *fusion bisimilar*, written $P \sim Q$, if and only if there exists a fusion bisimulation between them. The relation $\sim$ is the largest fusion bisimulation.

Another bisimulation worth exploring is the standard strong bisimulation, which requires that fusion transitions match exactly. This bisimulation is a congruence and contained in the fusion bisimulation. We do not know whether the containment is strict. This question relates to an open problem for the π -calculus without replication or summation, of whether strong bisimulation is closed with respect to substitution.

The Structured LTS

Our goal is to show that the fusion bisimulation in Definition 8 is a congruence. However, although the quotiented LTS of Figure 2 is simple due to the presence of the structural congruence rule, the same rule is a problem for proofs. We therefore introduce a *structured* LTS, in which the structural congruence rule is replaced. This structured LTS is ultimately used in Theorem 9 to prove that bisimulation is a congruence. The power of the structured LTS is that we can analyse the transition $P \xrightarrow{\alpha}_s Q$ by looking at the structure of P and the label α .

The structured LTS is given in Figure 3. Note the first fusion rule. It allows us to deduce for example that $\langle x=y \rangle | x.P$ can undergo the transition $\xrightarrow{y}_s$ as well as $\xrightarrow{x}_s$.

We write $\sim_s$ for the bisimulation generated by the structured LTS, defined in the same way as for the quotiented LTS in Definition 8.

Theorem 9.

1. $P \sim_s Q$ implies $C[P] \sim_s C[Q]$.
2. $\sim = \sim_s$

From Theorem 9 we deduce the main result of this section: that the fusion bisimulation $\sim$ for the quotiented LTS is a congruence.

Towards Full Abstraction for the Fusion Calculus

We believe that hyper-equivalence for the fusion calculus [10] corresponds to open bisimulation for its embedding in the π_F -calculus. The following examples illustrate labelled transitions in the fusion calculus on the left, and the corresponding transitions in the π_F -calculus on the right:

$$\begin{array}{ll}
 \bar{u}x.P \xrightarrow{\bar{u}x}_{fu} P & \bar{u}.(\langle x \rangle | P^*) \xrightarrow{\bar{u}}_{\pi_F} \langle x \rangle | P^* \\
 (\nu x)\bar{u}x.P \xrightarrow{(x)\bar{u}x}_{fu} P & (\nu x)\bar{u}.(\langle x \rangle | P^*) \xrightarrow{\bar{u}}_{\pi_F} (\nu x)(\langle x \rangle | P^*) \\
 \bar{u}x.P \mid uy.Q \xrightarrow{x=y}_{fu} P \mid Q & \bar{u}.(\langle x \rangle | P^*) \mid u.(\langle y \rangle | Q^*) \xrightarrow{\tau}_{\pi_F} \langle x=y \rangle | P^* \mid Q^*
 \end{array}$$

First consider the transitions for the fusion calculus. The labels $\bar{u}x$ and $(\nu x)\bar{u}x$ are standard. The label $x=y$ states that a fusion has occurred as a consequence of a reaction. Notice that it is not the same as the label $?x=y$ in the π_F -calculus, which states that an external fusion must be present for reaction to occur. Now compare the transitions of the fusion calculus with those of the π_F -calculus. The additional information conveyed by a fusion calculus label, is conveyed in the π_F -calculus by the interface of the resulting process.

Victor and Parrow show that hyper-equivalence does not correspond to open bisimulation for the π -calculus [10]. The same result holds for the π_F -calculus with replication. The difference is illustrated by the process $(\nu x)(\bar{u}.(\langle xy \rangle | P))$. In the π -calculus the names x and y can never be substituted for equal names. In the π_F -calculus they can, using the context $- \mid u.(\langle zz \rangle)$.

5 Conclusions

Several calculi with name-fusions have recently been proposed. These include the fusion calculus [10], the related chi calculus [1] and the π_I -calculus [11]. In all these calculi the fusions occur *implicitly* in the reaction relation. With the π_F -calculus we have introduced *explicit* fusions. Explicit fusions are processes which can exist in parallel with other processes. They are at least as expressive as implicit fusions. The effect of explicit fusions is described by the structural congruence, not by the reaction relation. The simplicity of the π_F -calculus follows directly from its use of explicit fusions.

We have given embedding results for the π -calculus and the fusion calculus in the π_F -calculus. The embedding for the fusion calculus is weaker than that for the π -calculus.

This is to be expected. The π_F -reaction is a local reaction between input and output processes, whose result contains explicit fusions. In contrast, reaction in the fusion calculus has the side-condition that certain names be restricted. The effect of this is to permit only those reactions which do not result in explicit fusions. This is why explicit fusions are not used (or needed) in the fusion calculus.

We have presented a bisimulation congruence for the π_F -calculus. We believe that hyper-equivalence for the fusion calculus is the same as the bisimulation arising from its embedding in the π_F -calculus.

Ongoing Research

Our work on explicit fusions originally arose from a study of process frameworks. We have developed a framework based on the structural congruence studied here [4,2]. It is related to the action calculus framework of Milner [9,3]. Explicit fusions allow us to work in a process algebra style, rather than the categorical style used for action calculi.

We are currently exploring an embedding of the λ -calculus in the π_F -calculus. Explicit fusions allow for a translation that is purely compositional, unlike the analogous translations into the π -calculus and fusion calculus. It remains further work to relate behavioural congruence for the λ -calculus with the bisimulation arising from its embedding in the π_F -calculus.

Acknowledgements. We thank Peter Sewell, Robin Milner and the anonymous referees. Gardner is supported by an EPSRC Advanced Fellowship, and Wischik by an EPSRC Studentship.

References

1. Y. Fu. Open bisimulations on chi processes. In *CONCUR*, LNCS 1664. Springer, 1999.
2. P. Gardner. From process calculi to process frameworks. In *CONCUR*, 2000. To appear.
3. P. Gardner and L. Wischik. Symmetric action calculi (abstract). Manuscript online.
4. P. Gardner and L. Wischik. A process framework based on the π_F calculus. In *EXPRESS*, volume 27. Elsevier Science Publishers, 1999.
5. K. Honda. Elementary structures in process theory (1): sets with renaming. *Mathematical Structures in Computer Science*. To appear.
6. K. Honda and N. Yoshida. On reduction-based process semantics. In *Foundations of Software Technology and Theoretical Computer Science*, LNCS 761. Springer, 1993.
7. M. Merro. On equators in asynchronous name-passing calculi without matching. In *EXPRESS*, volume 27. Elsevier Science Publishers, 1999.
8. R. Milner. *Communicating and mobile systems: the pi calculus*. CUP, 1999.
9. Robin Milner. Calculi for interaction. *Acta Informatica*, 33(8), 1996.
10. J. Parrow and B. Victor. The fusion calculus: expressiveness and symmetry in mobile processes. In *LICS*. IEEE, Computer Society Press, 1998.
11. D. Sangiorgi. Pi-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(2), 1996.
12. D. Sangiorgi. A theory of bisimulation for the π -calculus. *Acta Informatica*, 33, 1996.
13. B. Victor and J. Parrow. Concurrent constraints in the fusion calculus. In *ICALP*, LNCS 1443. Springer, 1998.
14. L. Wischik. 2001. Ph.D. thesis. In preparation.

State Space Reduction Using Partial τ -Confluence

Jan Friso Groote^{1,2} and Jaco van de Pol¹

¹ CWI, P.O.-box 94.079, 1090 GB Amsterdam, The Netherlands

² Department of Mathematics and Computing Science

Eindhoven University of Technology

P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Abstract. We present an efficient algorithm to determine the maximal class of confluent τ -transitions in a labelled transition system. Confluent τ -transitions are inert with respect to branching bisimulation. This allows to use τ -priorisation, which means that in a state with a confluent outgoing τ -transition all other transitions can be removed, maintaining branching bisimulation. In combination with the removal of τ -loops, and the compression of τ -sequences this yields an efficient algorithm to reduce the size of large state spaces.

1 Introduction

A currently common approach towards the automated analysis of distributed systems is the following. Specify an instance of the system with a limited number of parties and a small data domain. Subsequently, generate the state space of this system and reduce it using an appropriate equivalence, for which weak bisimulation [16] or branching bisimulation [6] generally serves quite well. The reduced state space can readily be manipulated, and virtually all questions about it can be answered with ease, using appropriate, available tools (see e.g. [5,4,8] for tools to generate and manipulate state spaces). By taking the number of involved parties and the data domains as large as possible, a good impression of the behaviour can be obtained and many of its problems are exposed, although total correctness cannot be verified in general.

A problem of the sketched route is that the state spaces that are generated are as large as possible, which, giving the growing memory capacities of contemporary computers is huge. So, as the complexity of reduction algorithms is generally more than linear, the time required to reduce these state spaces increases even more. Let n be the number of states and m be the number of transitions of a state space. The time complexity of computing the minimal branching bisimilar state space is $\mathcal{O}(nm)$ [11]; for weak bisimulation this is $\mathcal{O}(n^\alpha)$ where $\alpha \approx 2.376$ is the constant required for matrix multiplication [12].

We introduce a state space reduction algorithm of complexity $\mathcal{O}(m \text{Fanout}_\tau^3)$ where Fanout_τ is the maximal number of outgoing τ -transitions of a node in

⁰ A technical report including full proofs appeared as [9].

E-mail: JanFriso.Groote@cwi.nl, Jaco.van.de.Pol@cwi.nl

the transition system. Assuming that for certain classes of transition systems $Fanout_\tau$ is constant, our procedure is linear in the size of the transition system.

The reduction procedure is based on the detection of τ -confluence. Roughly, we call a τ -transition from a state s confluent if it commutes with any other a -transition starting in s . When the maximal class of confluent τ -transitions has been determined, τ -priorisation is applied. This means that any outgoing confluent τ -transition may be given “priority”. In some cases this reduces the size of the state space with an exponential factor. For convergent systems, this reduction preserves branching bisimulation, so it can serve as a preprocessing step to computing the branching bisimulation minimization.

Related Work. Confluence has always been recognized as an important feature of the behaviour of distributed communicating systems. In [16] a chapter is devoted to various notions of determinacy of processes, among which confluence, showing that certain operators preserve confluence, and showing how confluence can be used to verify certain processes. In [14,19] these notions have been extended to the π -calculus. In [10] an extensive investigation into various notions of global confluence for processes is given, where it is shown that by applying τ -priorisation, state spaces could be reduced substantially. In particular the use of confluence for symbolic verification purposes in the context of linear process operators was discussed. In [17] it is shown how using a typing system on processes it can be determined which actions are confluent, without generating the transition system. In [13] such typing schemes are extended to the π -calculus. Our method is also strongly related to partial order reductions [7,21], where an independence relation on actions and a property to be checked are assumed. The property is used to hide actions, and the independence relation is used for a partial order reduction similar to our τ -priorisation.

Our primary contribution consists of providing an algorithm that determines the maximal set of confluent τ -transitions for a given transition system. This differs from the work in [10] which is only applicable if all τ -transitions are confluent, which is often not the case. It also differs from approaches that use type systems or independence relations, in order to determine a subset of the confluent τ -transitions. These methods are incapable of determining the maximal set of confluent τ -transitions in general.

In order to assess the effectiveness of our state space reduction strategy, we implemented it and compared it to the best implementation of the branching bisimulation reduction algorithm that we know [3]. In combination with τ -loop elimination, and τ -compression, we found that in the worst case the time that our algorithm required was in the same order as the time for the branching bisimulation algorithm. Under the favourable conditions that there are many equivalence classes and many visible transitions, our algorithm appears to be a substantial improvement over the branching bisimulation reduction algorithm.

Acknowledgements. We thank Holger Hermanns for making available for comparison purposes a new implementation of the branching bisimulation algorithm devised by him and others [3].

2 Preliminaries

In this section we define elementary notions such as labelled transition systems, confluence, branching bisimulation and T -convergence.

Definition 1. A labelled transition system is a triple $A = (S, Act, \longrightarrow)$ where S is a set of states; Act is a set of actions; and $\longrightarrow \subseteq S \times Act \times S$ is the transition relation. We assume that $\tau \in Act$, a special constant denoting internal action.

We write $\xrightarrow{a}$ for the binary relation $\{\langle s, t \rangle \mid \langle s, a, t \rangle \in \longrightarrow\}$. We write $s \xrightarrow{\tau^*} t$ iff there is a sequence $s_0, \dots, s_n \in S$ with $n \geq 0$, $s_0 = s$, $s_n = t$ and $s_i \xrightarrow{\tau} s_{i+1}$. We write $t \rightleftharpoons s$ iff $t \xrightarrow{\tau^*} s$ and $s \xrightarrow{\tau^*} t$, i.e. s and t lie on a τ -loop. Finally, we write $s \xrightarrow{\bar{a}} s'$ if either $s \xrightarrow{a} s'$, or $s = s'$ and $a = \tau$.

A set $T \subseteq \xrightarrow{\tau}$ is called a *silent transition set* of A . We write $s \xrightarrow{\tau}_T t$ iff $\langle s, t \rangle \in T$. With $s \xrightarrow{\bar{\tau}}_T t$ we denote $s = t$ or $s \xrightarrow{\tau}_T t$. We define the set $Fanout_\tau(s)$ for a state s by: $Fanout_\tau(s) = \{s' \xrightarrow{\tau} s' \mid s' \in S\}$. A is *finite* if S and Act have a finite number of elements. In this case, n denotes the number of states of A , m is the number of transitions in A and m_τ denotes the number of τ -transitions. Furthermore, we write $Fanout_\tau$ for the maximal size of the set $Fanout_\tau(s)$.

Definition 2. Let $A = (S, Act, \longrightarrow)$ be a labelled transition system and T be a silent transition set of A . We call A *T -confluent* iff for each transition $s \xrightarrow{\tau}_T s'$ and for all $s \xrightarrow{a} s''$ ($a \in Act$) there exists a state $s''' \in S$ such that $s' \xrightarrow{\bar{a}} s'''$, $s'' \xrightarrow{\bar{\tau}}_T s'''$. We call A *confluent* iff A is $\xrightarrow{\tau}$ -confluent.

Definition 3. Let $A = (S_A, Act, \longrightarrow_A)$ and $B = (S_B, Act, \longrightarrow_B)$ be labelled transition systems. A relation $R \subseteq S_A \times S_B$ is called a *branching bisimulation relation* on A and B iff for every $s \in S_A$ and $t \in S_B$ such that sRt it holds that

1. If $s \xrightarrow{a}_A s'$ then for some t' and t'' , $t \xrightarrow{\tau^*}_B t' \xrightarrow{\bar{a}}_B t''$ and sRt' and $s'Rt''$.
2. If $t \xrightarrow{a}_B t'$, then for some s' and s'' , $s \xrightarrow{\tau^*}_A s' \xrightarrow{\bar{a}}_A s''$ and $s'Rt$ and $s''Rt'$.

For states $s \in S_A$ and $t \in S_B$ we write $s \rightleftharpoons_b t$ on $A \times B$, and say s and t are *branching bisimilar*, iff there is a branching bisimulation relation R on A and B such that sRt . In this case, $\rightleftharpoons_b$ itself is the maximal branching bisimulation, and it is an equivalence relation. A transition $s \xrightarrow{\tau} s'$ is called *inert* iff $s \rightleftharpoons_b s'$.

Theorem 1. Let $A = (S, Act, \longrightarrow)$ be a labelled transition system and let T be a silent transition set of A . If A is T -confluent, every $s \xrightarrow{\tau}_T s'$ is inert.

Proof. (sketch) It can be shown that the relation $R = \xrightarrow{\bar{\tau}}_T$ is a branching bisimulation relation. $\square$

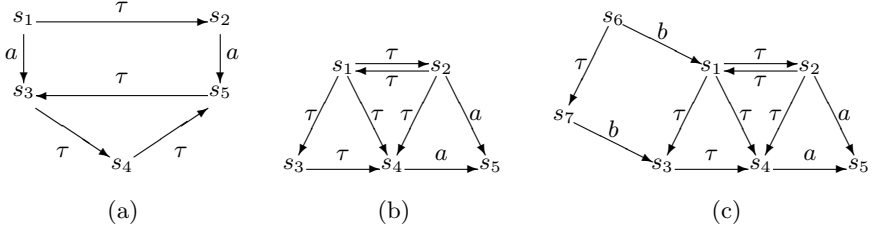


Fig. 1. Counterexamples to preservation of confluence

Lemma 1. *Let A be a labelled transition system. There exists a largest silent transition set T_{conf} of A , such that A is T_{conf} -confluent.*

Proof. (sketch) Consider the set $\mathcal{T}$, being the set of all silent transition sets T such that A is T -confluent. Define $T_{\text{conf}} = \bigcup \mathcal{T}$. $\square$

Definition 4. *Let $A = (S, \text{Act}, \longrightarrow)$ be a labelled transition system. We call A convergent iff there is no infinite sequence $s_1 \xrightarrow{\tau} s_2 \xrightarrow{\tau} \dots$.*

3 Elimination of τ -Cycles

In this section we define the removal of τ -loops from a transition system. The idea is to collapse each loop to a single state. This can be done, because $\rightleftharpoons$ is an equivalence relation on states.

Definition 5. *Let $A = (S, \text{Act}, \longrightarrow)$ be a labelled transition system. Define $[s]_A = \{t \in S \mid t \rightleftharpoons s\}$. Define the relation $[\longrightarrow]_A$, such that $S[\xrightarrow{a}]_A S'$ iff there exist $s \in S$, $s' \in S'$ such that $s \xrightarrow{a} s'$ but not $S = S'$ and $a = \tau$. Write $[S]_A$ for $\{[s]_A \mid s \in S\}$ and $[T]_A$ for the relation $\{[s]_A[\xrightarrow{a}]_A[t]_A \mid s \xrightarrow{a} t \in T\}$.*

Definition 6. *Let $A = (S, \text{Act}, \longrightarrow)$ be a labelled transition system. The τ -cycle reduction of A is the labelled transition system $A_{\otimes} = ([S]_A, \text{Act}, [\longrightarrow]_A)$.*

Using the algorithm to detect strongly connected components [1] it is possible to construct the τ -cycle reduction of a labelled transition system A in linear time.

Lemma 2. *Let $A = (S, \text{Act}, \longrightarrow)$ be a labelled transition system and let $A_{\otimes}$ be its τ -cycle reduction. Then for every state $s \in S$, $s \leftrightarrow_b [s]_A$ on $A \times A_{\otimes}$.*

We now show that taking the τ -cycle reduction can change the confluence structure of a process. Figure 1.a shows that a non-confluent τ -transition may become confluent after τ -cycle reduction: $s_1 \xrightarrow{\tau} s_2$ is not confluent before τ -cycle reduction, but it is afterwards. Conversely, Figure 1.b shows that a confluent τ -transition may become non-confluent after τ -cycle reduction. Observe that all τ -transitions are confluent. After τ -cycle reduction is applied, s_1 and s_2 are taken together, and we see that $\{s_1, s_2\} \xrightarrow{\tau}_{\otimes} \{s_3\}$ and $\{s_1, s_2\} \xrightarrow{a}_{\otimes} \{s_5\}$; but

there is no way to complete the diagram. We can extend the example slightly (Figure 1.c) showing that states that have outgoing confluent transitions before τ -cycle reduction, do not have these afterwards. Nevertheless, τ -cycle reduction is unavoidable in view of Example 1 (Section 5).

4 Algorithm to Calculate T_{conf}

We now present an algorithm to calculate T_{conf} for a given labelled transition system $A = (S, Act, \longrightarrow)$. First the required data structures and their initialization are described: Each transition $s \xrightarrow{\tau} s'$ is equipped with a boolean **candidate** that is initially set to true, indicating whether this transition is still a candidate to be put in T_{conf} . Furthermore, for every state s we store a list of all incoming transitions, as well as a list with outgoing τ -transitions. Also, each transition $s \xrightarrow{a} s'$ is stored in a hash table, such that given s , a and s' , it can be found in constant time, if it exists. Finally, there is a stack on which transitions are stored, in such a way that membership can be tested in constant time. Transitions on the stack must still be checked for confluence; initially, all transitions are put on the stack.

The algorithm now works as follows. As long as the stack is not empty, remove a transition $s \xrightarrow{a} s'$ from it. Check that $s \xrightarrow{a} s'$ is still confluent with respect to all τ -transitions outgoing from state s that have the variable **candidate** set. Checking confluence means that for each candidate transition $s \xrightarrow{\tau} s''$ it must be verified that either

- for some $s''' \in S$, $s'' \xrightarrow{a} s'''$ and $s' \xrightarrow{\tau} s'''$, which is still **candidate**;
- or $s'' \xrightarrow{a} s'$;
- or $a = \tau$ and $s' \xrightarrow{\tau} s''$ with the variable **candidate** set;
- or finally, $a = \tau$ and $s' = s''$.

For all transitions $s \xrightarrow{\tau} s''$ for which the confluence check with respect to some $s \xrightarrow{a} s'$ fails, the boolean **candidate** is set to false. If there is at least one transition $s \xrightarrow{\tau} s''$ for which the check fails, then all transitions $t \xrightarrow{a} s$ that are not on the stack must be put back on it. This can be done conveniently, using the list of incoming transitions of node s .

After the algorithm has terminated, i.e. when the stack is empty, the set T_{alg} is formed by all τ -transitions for which the variable **candidate** is still true. Termination of the algorithm follows directly from the following observation: either, the size of the stack decreases, while the number of candidate transitions remains constant; or the number of candidate transitions decreases, although in this case the stack may grow. Correctness of the algorithm follows from the theorem below, showing that $T_{alg} = T_{conf}$.

Lemma 3. *A is T_{alg} -confluent.*

Proof. Consider transitions $s \xrightarrow{a} s'$ and $s \xrightarrow{\tau}_T s''$. Consider the last step, with index say n , in the algorithm where $s \xrightarrow{a} s'$ is removed from the stack. The variable **candidate** of $s \xrightarrow{\tau} s''$ was never set to false, hence either:

- $a = \tau$ and $s' = s''$, or
- $s'' \xrightarrow{a} s'$, or
- $a = \tau$ and $s' \xrightarrow{\tau} s''$ with the variable **candidate** set (at step n), or
- for some $s''' \in S$, $s' \xrightarrow{\tau} s'''$ was a candidate at step n , and $s'' \xrightarrow{a} s'''$.

In the first two cases it is obvious that $s \xrightarrow{a} s'$ and $s \xrightarrow{\tau} s''$ are T_{alg} -confluent w.r.t. each other. In the last two cases confluence is straightforward, if respectively $s' \xrightarrow{\tau} s''$ or $s' \xrightarrow{\tau} s'''$ are still candidate transitions when the algorithm terminates. This means that these transitions are put in T_{alg} . If, however, this is not the case, then there is a step $n' > n$ in the algorithm where the **candidate** variable of $s' \xrightarrow{\tau} s''$ or $s' \xrightarrow{\tau} s'''$, respectively, has been reset. In this case each transition ending in s' is put back on the stack. In particular $s \xrightarrow{a} s'$ is put on the stack to be removed at some step $n'' > n' > n$, contradicting the fact that n was the last such step. $\square$

Theorem 2. $T_{conf} = T_{alg}$

Proof. From the previous lemma, it follows that $T_{alg} \subseteq T_{conf}$. We now prove the reverse. Assume towards a contradiction that $s \xrightarrow{\tau} s'$ in T_{conf} is the first transition in the algorithm, whose variable **candidate** is erroneously marked false. This only happens when confluence w.r.t. some $s \xrightarrow{a} s''$ fails. By T_{conf} -confluence, for some s''' , $s'' \xrightarrow{\bar{\tau}}_{T_{conf}} s'''$ and $s' \xrightarrow{\bar{a}} s'''$. As $s \xrightarrow{\tau} s'$ is marked false, it must be the case that $s'' \xrightarrow{\tau} s'''$, and its **candidate** bit has been reset earlier in the algorithm. But this contradicts the fact that we are considering the first instance where a boolean **candidate** was erroneously set to false. $\square$

Lemma 4. *The algorithm terminates in $\mathcal{O}(m \text{ Fanout}_\tau^3)$ steps.*

Proof. Checking that a transition $s \xrightarrow{a} s'$ is confluent, requires $\mathcal{O}(\text{Fanout}_\tau^2)$ steps: for each $\xrightarrow{\tau}$ -successor s'' of s we have to try all $\xrightarrow{\tau}$ -successors of s'' . This can be conveniently done using the list of outgoing τ -transitions from s' and s'' . The check whether $s'' \xrightarrow{a} s'''$ is a single hash table lookup.

Every transition $s \xrightarrow{a} s'$ is put at most $\text{Fanout}_\tau + 1$ times on the stack: initially and each time the variable **candidate** of a τ -successor of s' is reset. For m transitions this leads to the upper bound: $\mathcal{O}(m \text{ Fanout}_\tau^3)$. $\square$

Note that it requires $\mathcal{O}(m \text{ Fanout}_\tau^2)$ to check whether a labelled transition system is τ -confluent with respect to all its τ -transitions. As determining the set T_{conf} seems more difficult than determining global τ -confluence, and we only require a factor Fanout_τ to do so, we expect that the complexity of our algorithm cannot be improved. We have looked into establishing other forms of partial τ -confluence (cf. [10]), especially forms where, given $s \xrightarrow{a} s'$ and $s \xrightarrow{\tau} s''$, it suffices to find some state s''' such that $s' \xrightarrow{\tau^*} s'''$ and $s'' \xrightarrow{\tau^* a \tau^*} s'''$. However, doing this requires the dynamic maintenance of the transitive τ -closure relation, which we could not perform in a sufficiently efficient manner to turn it into an effective preprocessing step for branching bisimulation.

5 τ -Priorisation and τ -Compression

After the set T_{conf} for a labelled transition system A has been determined we can “harvest” by applying τ -priorisation and calculating a form of τ -compression. Both operations can be applied in linear time, and moreover, reduce the state space. The τ -priorisation operation allows to give precedence to silent steps, provided they are confluent. This is defined as follows:

Definition 7. Let $A = (S, Act, \longrightarrow_A)$ be a labelled transition system and let T be a set of τ -transitions of A . We say that a transition system $B = (S, Act, \longrightarrow_B)$ is a τ -priorisation of A with respect to T iff for all $s, s' \in S$ and $a \in Act$

- if $s \xrightarrow{a}_B s'$ then $s \xrightarrow{a}_A s'$, and
- if $s \xrightarrow{a}_A s'$ then $s \xrightarrow{a}_B s'$ or for some $s'' \in S$ it holds that $s \xrightarrow{\tau}_B s'' \in T$.

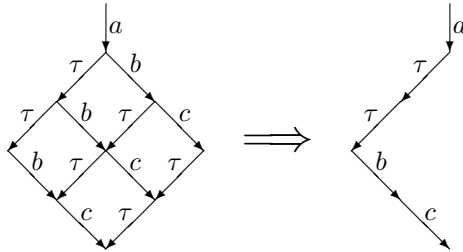
It holds that τ -priorisation maintains branching bisimulation.

Theorem 3. Let $A = (S, Act, \longrightarrow_A)$ be a convergent, labelled transition system, which is T -confluent for some silent transition set T . Let $B = (S, Act, \longrightarrow_B)$ be a τ -priorisation of A w.r.t. T . Then for each state $s \in S$, $s \dot{\leftrightarrow}_b s$ on $A \times B$.

Proof. (sketch) The auto-bisimulation $\dot{\leftrightarrow}_b^A$ on $A \times A$ is also a branching bisimulation relation on $A \times B$. This is proved using an auxiliary lemma, which ensures that if $s \dot{\leftrightarrow}_b^A t$, $s \xrightarrow{\tau^*}_A s' \xrightarrow{a}_A s''$ and $s \dot{\leftrightarrow}_b^A s'$, then for some t' and t'' , $t \xrightarrow{\tau^*}_B t' \xrightarrow{a}_B t''$, $t \dot{\leftrightarrow}_b^A t'$ and $s'' \dot{\leftrightarrow}_b^A t''$. This is proved by induction on $\xrightarrow{\tau}$. $\square$

Example 1. Convergence of a labelled transition system is a necessary precondition. Let a labelled transition system A be given with single state s and two transitions: $\{s \xrightarrow{\tau} s, s \xrightarrow{a} s\}$. It is clearly confluent, but not convergent. The τ -priorisation is a single τ -loop, which is not branching bisimilar with A .

The τ -priorisation w.r.t. a given set T of transitions can be computed in linear time, by traversing all states, and if there is an outgoing T -transition, removing all other outgoing transitions. Consider the labelled transition system below. All τ -transitions are confluent, and τ -priorisation removes more than half of the transitions.



A typical pattern in τ -prioritised transition systems are long sequences of τ -steps that can easily be removed. We call the operation doing so τ -compression.

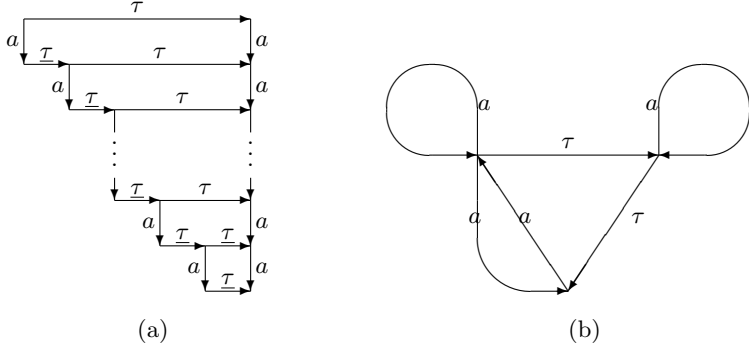


Fig. 2. The effect of repetition

Definition 8. Let $A = (S, Act, \longrightarrow)$ be a convergent labelled transition system. For each state $s \in S$ we define with well-founded recursion on $\xrightarrow{\tau}$ the τ -descendant of s , notation $\tau^*(s)$, as follows: $\tau^*(s) = \tau^*(s')$ if for some s' , $s \xrightarrow{\tau}_A s'$ is the only transition leaving s , and $\tau^*(s) = s$ otherwise. The τ -compression of A is the transition system $A_F = (S, Act, \longrightarrow_{A_F})$ where $\longrightarrow_{A_F} = \{\langle s, a, \tau^*(s') \rangle \mid s \xrightarrow{a}_A s'\}$.

Theorem 4. Let $A = (S, Act, \longrightarrow)$ be a labelled transition system and A_F the τ -compression of A . Then for all $s \in S$, $s \leftrightarrow_b s$ on $A \times A_F$.

Proof. (sketch) It can be shown that $R = \{\langle s, s \rangle \mid s \in S\} \cup \{\langle s, \tau^*(s) \rangle \mid s \in S\}$ is a branching bisimulation. $\square$

Note that the τ -compression can be calculated in linear time. During a depth first sweep $\tau^*(s)$ can be calculated for each state s . Then by traversing all transitions, each transition $s \xrightarrow{a} s'$ can be replaced by $s \xrightarrow{a} \tau^*(s')$.

6 The Full Algorithm and Benchmarks

In this section we summarize all operations in a complete algorithm and give some benchmarks to indicate its usefulness. Note that τ -compression can make new diagrams confluent (in fact we discovered this by experimenting with the implementation). Therefore, we present our algorithm as a fixed point calculation, starting with a transition system $A = (S, Act, \longrightarrow)$:

```

B := A⊗;
repeat
  n := #states of B;
  calculate Tconf for B;
  apply  $\tau$ -priorisation to B w.r.t. Tconf;
  apply  $\tau$ -compression to B;
while n ≠ #states of B;

```

The example in Figure 2.a shows that in the worst case the loop in the algorithm must be repeated $\Omega(n)$ times for a labelled transition system with n states. Only the underlined τ -transitions are confluent. Therefore, each subsequent iteration of the algorithm removes the bottom tile, connecting the two arrows in the one but last line.

An improvement, which we have not employed, might be to apply strong bisimulation reductions, in this algorithm. As shown by [18] strong bisimulation can be calculated in $\mathcal{O}((m+n) \log n)$, which although not being linear, is quite efficient. Unfortunately, Figure 2.b shows an example which is minimal with respect to all mentioned reductions and strong bisimulation, but not with respect to branching bisimulation.

In order to understand the effect of partial confluence checking we have applied our algorithm to a number of examples. We can conclude that if the number of internal steps in a transition system is relatively low, and the number of equivalence classes is high, our algorithm performs particularly well compared to the best implementation [3] of the standard algorithm for branching bisimulation [11]. Under less favourable circumstances, we see that the performance is comparable with the implementation in [3].

In Table 1 we summarize our experience with 5 examples. In the rows we list the number of states “ n ” and transitions “ m ” of each example. The column under “ n (red1)” indicates the size of the state space after 1 iteration of the algorithm. “#iter” indicates the number of iterations of the algorithm to stabilize, and the number of states of the resulting transition system is listed under “ n (red tot)”. The time to run the algorithm for partial confluence checking is listed under “time conf”. The time needed to carry out branching bisimulation reduction and the size of the resulting state space are listed under “time branch” and “ n min”, respectively. The local confluence reduction algorithm was run on a SGI Powerchallenge with a 300MHz R12000 processor. The branching bisimulation reduction algorithm was run on a 300MHz SUN Ultra 10.

The Firewire benchmark is the firewire or IEEE 1394 link protocol with 2 links and a bus as described in [15,20]. The SWP1.2 and SWP1.3 examples are sliding window protocols with window size 1, and with 2 and 3 data elements, respectively. The description of this protocol can be found in [2]. The processes PAR2.12 and PAR6.7 are specially constructed processes to see the effect of the relative number of τ -transitions and equivalence classes on the branching bisimulation and local confluence checking algorithms. They are defined as follows:

$$\text{PAR2.12} = \prod_{i=1}^{12} \tau a_i \quad \text{PAR6.7} = \prod_{i=1}^7 \tau a_i b_i c_i d_i e_i$$

Note that in these cases, partial confluence checking finds a minimal state space w.r.t. branching bisimulation.

Table 1. Benchmarks showing the effect of partial τ -confluence checking

	n	m	n (red1)	#iter	n (red tot)	time conf	n min	time branch
Firewire	372k	642k	46k	4	23k	3.6s	2k	132s
SWP1.2	320k	1.9M	32k	6	960	13s	49	9s
SWP1.3	1.2M	7.5M	127k	6	3k	57s	169	136s
PAR2.12	531k	4.3M	4k	2	4k	98s	4k	64s
PAR6.7	824k	4.9M	280k	2	280k	55s	280k	369s

References

1. A.V. Aho, J.E. Hopcroft and J.D. Ullman. Data structures and algorithms. Addison-Wesley, 1983.
2. M.A. Bezem and J.F. Groote. A correctness proof of a one bit sliding window protocol in μ CRL. The Computer Journal, 37(4): 289-307, 1994.
3. M. Cherif and H. Garavel and H. Hermanns. The bcg_min user manual, version 1.1. http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html, 1999.
4. D. Dill, C.N. Ip and U. Stern. Murphi description language and verifier. <http://sprout.stanford.edu/dill/murphi.html>, 1992-2000.
5. H. Garavel and R. Mateescu. The Caesar/Aldebaran development package. <http://www.inrialpes.fr/vasy/cadp/>, 1996-2000.
6. R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. In *Journal of the ACM*, 43(3):555-600, 1996.
7. P. Godefroid and P. Wolper. A partial approach to model checking. *Information and Computation*, 110(2):305-326, 1994.
8. J.F. Groote and B. Lissers. The μ CRL toolset. <http://www.cwi.nl/~mcrl>, 1999-2000.
9. J.F. Groote and J.C. van de Pol. *State space reduction using partial τ -confluence*. Technical Report CWI-SEN-R0008, March 2000. Available via <http://www.cwi.nl/~vdpol/papers/>.
10. J.F. Groote and M.P.A. Sellink. Confluence for process verification. In *Theoretical Computer Science B*, 170(1-2):47-81, 1996.
11. J.F. Groote and F.W. Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In *Proc. 17th ICALP*, LNCS 443, 626-638. Springer-Verlag, 1990.
12. P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43-68, 1990.
13. N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the π -calculus. In: *Proceedings of the 23rd POPL*, pages 358-371. ACM press, January 1996.
14. X. Liu and D. Walker. Confluence of processes and systems of objects. In *Proceedings of TAPSOFT'95*, pages 217-231, LNCS 915, 1995.
15. S.P. Luttik. Description and formal specification of the link layer of P1394. Technical Report SEN-R9706, CWI, Amsterdam, 1997.
16. R. Milner. *Communication and Concurrency*. Prentice Hall International, 1989.
17. U. Nestmann and M. Steffen. Typing confluence. In: *Proceedings of FMICS'97*, pages 77-101. CNR Pisa, 1997.

18. R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973-989, 1987.
19. A. Philippou and D. Walker. On confluence in the pi-calculus. *24th Int. Coll. on Automata, Languages and Programming*, LNCS 1256, Springer-Verlag, 1997.
20. M. Sighireanu and R. Mateescu. Verification of the link layer protocol of the IEEE-1394 serial bus (firewire): an experiment with E-LOTOS. In *Journal on Software Tools for Technology Transfer (STTT)*, 2(1):68-88, 1998.
21. A. Valmari. A stubborn attack on state explosion. In *Proc. of Computer Aided Verification*, LNCS 531, pages 25-42, Springer-Verlag, 1990.

Reducing the Number of Solutions of NP Functions^{*}

Lane A. Hemaspaandra¹, Mitsunori Ogihara¹, and Gerd Wechsung²

¹ Department of Computer Science, University of Rochester
Rochester, NY 14627, USA

² Institut für Informatik, Friedrich-Schiller-Universität Jena
07743 Jena, Germany

Abstract. We study whether one can prune solutions from NP functions. Though it is known that, unless surprising complexity class collapses occur, one cannot reduce the number of accepting paths of NP machines [17], we nonetheless show that it often is possible to reduce the number of solutions of NP functions. For finite cardinality types, we give a sufficient condition for such solution reduction. We also give absolute and conditional necessary conditions for solution reduction, and in particular we show that in many cases solution reduction is impossible unless the polynomial hierarchy collapses.

1 Introduction and Discussion

Let $\text{NP}_{\text{N+V}}$ denote the set of all (possibly partial, possibly multivalued) functions computable by nondeterministic polynomial-time Turing machines. That is, such a function f will map from strings x to the set $\{z \mid \text{some accepting path of } M(x) \text{ has } z \text{ as its output (i.e., as a “solution”)}\}$. $\text{NP}_{\text{N+V}}$ functions, known in the literature as NPMV (“nondeterministic polynomial-time (potentially) multivalued”) functions, have been extensively studied since they were introduced in the 1970s by Book, Long, and Selman ([1,2], see also [19]).

Much of this study recently has focused on the issue of whether even NP functions can prune solutions away from NP functions. As Naik, Rogers, Royer, and Selman [15] have elegantly pointed out, the motivation for this is multifold: in the broadest sense this addresses the central complexity-theoretic notion of measuring how resources (such as allowed output cardinality) enable computation, more specifically this addresses the power of nondeterminism, and more specifically still this issue is deeply tied ([20,12], see also [6,11]) to NP-search functions and the complexity of function inversion. Also worth contrasting with this paper’s proof that number of solutions of NP functions can be reduced in

^{*} Email: {lane,ogihara}@cs.rochester.edu, wechsung@informatik.uni-jena.de. Supported in part by grants DARPA-F30602-98-2-0133, NSF-CCR-9701911, NSF-INT-9726724, and NSF-INT-9815095/DAAD-315-PPP-gü-ab. Work done in part while the first author was visiting Friedrich-Schiller-Universität Jena and Julius-Maximilians-Universität Würzburg, and while the third author was visiting RIT.

various ways is the fact, due to Ogiwara and Hemachandra [17], that (unless surprising complexity class collapses occur) one cannot in general reduce even by one (proper decrement) the number of accepting paths of NP machines.

To discuss rigorously whether NP functions can prune solutions from NP functions, we need a formal way to capture this. The notion of refinement exactly captures this, and is used in the literature for exactly this purpose. Given (possibly partial, possibly multivalued) functions f and f' , we say that f' is a refinement (see for example the excellent survey by Selman [20]) of f if for each $x \in \Sigma^*$, (1) $f'(x)$ has at least one solution iff $f(x)$ has at least one solution, and (2) each solution of $f'(x)$ is a solution of $f(x)$. Given any two function classes $\mathcal{C}_1$ and $\mathcal{C}_2$, we say that $\mathcal{C}_1 \subseteq_c \mathcal{C}_2$ (" $\mathcal{C}_1$ functions always have $\mathcal{C}_2$ refinements") if for each function $f \in \mathcal{C}_1$ there is a function $f' \in \mathcal{C}_2$ such that f' is a refinement of f .

For any $A \subseteq \mathbb{N}^+$, NP_AV denotes the class of all $\text{NP}_{\mathbb{N}^+}\text{V}$ functions f satisfying $(\forall x \in \Sigma^*)[\text{the number of solutions of } f(x) \text{ is an element of } \{0\} \cup A]$.

Surprisingly, for the first twenty years after the classes $\text{NP}_{\mathbb{N}^+}\text{V}$ and $\text{NP}_{\{1\}}\text{V}$ (referred to in the literature respectively as NPMV and NPSV: nondeterministic polynomial-time {multivalued, single-valued} functions) were defined, there was no evidence against the dramatic possibility that $\text{NP}_{\mathbb{N}^+}\text{V} \subseteq_c \text{NP}_{\{1\}}\text{V}$, i.e., that all multivalued NP functions have single-valued NP refinements. (This is known to be equivalent to the claim that there is an NP function that on each satisfiable boolean formula as input finds exactly one satisfying assignment.) In the 1990s, Hemaspaandra, Naik, Ogiwara, and Selman [9] finally gave concrete evidence against this by proving the following result.

Theorem 1. [9] *If $\text{NP}_{\mathbb{N}^+}\text{V} \subseteq_c \text{NP}_{\{1\}}\text{V}$ then $\text{PH} = \text{ZPP}^{\text{NP}}$.*

Thus, if the polynomial hierarchy does not collapse, the following remarkable state holds: NP functions can find all satisfying assignments of boolean formulas but cannot find (exactly) one satisfying assignment to boolean formulas [9]. (Though it would be impossible for such a claim to hold for deterministic computation, as there finding one solution is provably no harder than finding all solutions, for nondeterministic computation this state is neither impossible nor paradoxical, though the fact that finding all solutions is simpler than finding one solution may at first be disconcerting.)

In fact, Hemaspaandra et al. proved something a bit stronger than Theorem 1.

Theorem 2. [9] *$\text{NP}_{\{1,2\}}\text{V} \subseteq_c \text{NP}_{\{1\}}\text{V}$ only if $\text{PH} = \text{ZPP}^{\text{NP}}$.*

Building on this, Ogiwara [16] and Naik et al. [15] showed that from weaker hypotheses one could reach weaker conclusions that nonetheless are strong enough to cast strong doubt on their hypotheses.

Theorem 3. [16] *For each k , $0 < k < 1$, if $\text{NP}_{\mathbb{N}^+}\text{V} \subseteq_c \text{NP}_{\{1, \dots, n^k\}}\text{V}$ then $\text{PH} = \text{NP}^{\text{NP}}$.¹*

¹ Note here that n in $\text{NP}_{\{1, \dots, n^k\}}\text{V}$ is the input length, so this statement is of different formal quality than those we study in this paper.

Theorem 4. [15] *For each $k \geq 1$, if $\text{NP}_{\{1, \dots, k+1\}} V \subseteq_c \text{NP}_{\{1, \dots, k\}} V$ then $\text{PH} = \text{NP}^{\text{NP}}$.*

Theorems 1, 2, 3, and 4 say that, for the cases they cover, one cannot prune solutions unless the polynomial hierarchy collapses.

However, note that all these theorems cover cases in which the allowed non-zero solution cardinalities form a (finite or infinite) prefix of $\{1, 2, 3, \dots\}$. That is, the theorems deal just with the question: Given any NP function having on each input at most ℓ solutions (ℓ is either ∞ or an element of $\mathbb{N}^+$) will it always be the case that there exists another NP function that is a refinement of the first function and that on each input has at most ℓ' solutions ($\ell', \ell' < \ell$, is an element of $\mathbb{N}^+$). In fact, prior to the present paper only such “left-prefix-of- $\mathbb{N}^+$ ”-cardinality sets had been studied.

We introduce the notion $\text{NP}_A V$ and propose as natural the following general challenge.

Challenge 5. *Completely characterize, perhaps under some complexity-theoretic assumption, the sets $A \subseteq \mathbb{N}^+$, and $B \subseteq \mathbb{N}^+$ such that $\text{NP}_A V \subseteq_c \text{NP}_B V$.*

This question captures far more fully the issue of what types of cardinality reduction are generally possible via refinement of NP functions. Further, this also parallels the way language classes have been defined in complexity theory. There, notions of “acceptance types” and “promises about numbers of accepting paths” are natural. In fact, a language notion, “ NP_A ” can be found in the literature [4], and unifies many notions of counting-based acceptance (and see more generally the notion of leaf languages [3,21]). In our function case, we view the A of $\text{NP}_A V$ as a *cardinality type* since it specifies the allowed nonzero numbers of solutions.

Challenge 5 is very broad and ambitious, as it goes well beyond the cases considered in Theorems 1, 2, 3, and 4. The present paper focuses on the case of finite cardinality types— $\text{NP}_A V$ for sets $A \subseteq \mathbb{N}^+$ satisfying $\|A\| < \infty$. Section 2 presents a condition, for sets $A, B \subseteq \mathbb{N}^+$, $\|A\| < \infty$, $\|B\| < \infty$, sufficient to ensure $\text{NP}_A V \subseteq_c \text{NP}_B V$. This condition is not a complexity-theoretic assumption but rather is a simple statement about the sets A and B . *Thus, we will see that in many cases solution reduction is possible for NP functions, in contrast to Theorems 1, 2, 3, and 4 and in contrast to the known result [17] that unless shocking complexity class collapses occur accepting-path-cardinality reduction is not in general possible for NP machines.*

We conjecture that for finite cardinality types our sufficient condition is necessary unless the polynomial hierarchy collapses. Though we cannot prove that, Section 3 establishes broad necessary conditions for solution reduction under the assumption that the polynomial hierarchy does not collapse. These conditions subsume the previously known cases obtained in Hemaspaandra et al. and Naik et al. We also prove an absolute necessary condition, but we show that proving any sufficiently broad absolute necessary condition would immediately yield a proof that $\text{NP} \neq \text{coNP}$.

Section 4 revisits Theorem 4, which says that $\text{NP}_{\{1, \dots, k+1\}} V \subseteq_c \text{NP}_{\{1, \dots, k\}} V$ implies $\text{PH} = \text{NP}^{\text{NP}}$. Of course, most complexity researchers, deep down, believe

that $\text{NP}_{\{1, \dots, k+1\}} V \not\subseteq_c \text{NP}_{\{1, \dots, k\}} V$. If this belief is a correct guess about the state of the world, then Theorem 4 tells us nothing, as it is of the form “false $\implies \dots$.” Intuitively, one would hope that Theorem 4 is a reflection of some structural simplicity property of sets. Section 4 proves that this is indeed the case, via showing, along with an even broader result, that all NP sets that are $(k+1)$ -selective via $\text{NP}_{\{1, \dots, k\}} V$ functions in fact belong to the second level of Schöning’s low hierarchy [18]. Section 5 provides a more unified strengthening.

2 A Sufficient Condition

We now state our sufficient condition. Intuitively, one can think of this as a “narrowing-gap” condition as what it says is that the gaps² between the cardinalities in A and certain of the cardinalities in B have to form a (perhaps nonstrictly) decreasing sequence. Due to the page limit we omit the proof here. An interested reader may refer to the full paper [10].

Theorem 6. *For each pair of finite sets $A, B \subseteq \mathbb{N}^+$, $A = \{a_1, \dots, a_m\}$ with $a_1 < a_2 < \dots < a_m$, we have $\text{NP}_A V \subseteq_c \text{NP}_B V$ if*

$$\|A\| = 0 \text{ or } (\exists b_1, \dots, b_m \in B)[a_1 - b_1 \geq \dots \geq a_m - b_m \geq 0].$$

3 Necessary Conditions

We conjecture that for finite cardinality types the “narrowing gap” sufficient condition from Theorem 6 is in fact necessary unless the polynomial hierarchy collapses.

Narrowing-Gap Conjecture. *For each pair of finite sets $A \subseteq \mathbb{N}^+$ and $B \subseteq \mathbb{N}^+$ that do not satisfy the condition of Theorem 6, we have:*

$$\text{NP}_A V \subseteq_c \text{NP}_B V \implies \text{PH} = \text{NP}^{\text{NP}}.$$

Why didn’t we make an even stronger version of the conjecture that asserts that for finite cardinality types the condition of Theorem 6 is (unconditionally) necessary? After all, certain finite cardinality types violating the condition of Theorem 6 trivially do not allow solution reduction, as the following result shows.

Proposition 1. *Let $A \subseteq \mathbb{N}^+$, $B \subseteq \mathbb{N}^+$, $A \neq \emptyset$, and $B \neq \emptyset$. If $\min\{i \mid i \in A\} < \min\{i \mid i \in B\}$ then $\text{NP}_A V \not\subseteq_c \text{NP}_B V$.*

² “Gap” is used here in its common-language sense of “differences between integers” (that here happen to represent cardinalities of outputs) rather than in its term-of-art complexity-theoretic sense of differences between integers representing cardinalities of accepting paths.

Proof. Let A and B as in the hypothesis. Let $m = \min\{i \mid i \in A\}$. Let f be the function that maps from each $x \in \Sigma^*$ to the numbers $\{1, \dots, m\}$. Clearly, $f \in \text{NP}_A V$. Since f has exactly m outputs on each input, for any function g to be a refinement of f , $g(x)$ has to have output cardinality between 1 and m for every $x \in \Sigma^*$. However, this is not possible for any function in $\text{NP}_B V$ since $\min\{i \mid i \in B\} > m$. $\square$

Nonetheless, unconditionally showing that the condition of Theorem 6 is necessary for finite accepting types seems out of reach. The reason is that, due to the following result, showing the condition to be necessary would in fact prove that $\text{NP} \neq \text{coNP}$.

Theorem 7. *If $\text{NP} = \text{coNP}$ then for any set $A \subseteq \mathbb{N}^+$ it holds that $\text{NP}_A V \subseteq_c \text{NP}_{\{1\}} V$.*

In contrast, the Narrowing-Gap Conjecture does not seem to imply $\text{NP} \neq \text{coNP}$, or any other unexpected fact, in any obvious way. We suggest that the Narrowing-Gap Conjecture is a plausible long-term goal.

The following result shows that, if $\text{PH} \neq \text{NP}^{\text{NP}}$, then a wide range of cardinality types A and B do not have solution reduction.

Theorem 8. *Let $A, B \subseteq \mathbb{N}^+$ be nonempty. Suppose there exist four integers $c > 0$, $d > 0$, $e \geq 0$, and $\delta \geq 0$ satisfying the following conditions:*

- $d \leq c \leq 2d$ and $\delta < 2d - c$,
- $c, 2d + e \in A$,
- $c - \delta \geq \min\{i \mid i \in B\}$, and
- $2d - (2\delta + 1) \geq \max\{i \in B \mid i \leq 2d + e\}$.

Then $\text{NP}_A V \subseteq_c \text{NP}_B V$ implies $\text{PH} = \text{NP}^{\text{NP}}$.

Theorem 8 is a rather complex necessary condition, as it is loaded with degrees of freedom to let it be broad. Nonetheless, there are some cases it misses, for example due to the fact that the $d \leq c \leq 2d$ clause can limit us when dealing with certainly cardinality types with widely varying values. For example, regarding cardinality-2 cardinality types, Theorem 8 yields as a corollary result 1 below. However, we can also prove Theorem 9, which is another necessary-condition theorem and which seemingly does not follow (in any obvious way) from Theorem 8. We will prove Theorem 9 in Section 5.

Corollary 1. *For any integers $k > 0$, $k' > 0$, $k' \geq k$, if $\text{NP}_{\{k-1, k'\}} V \subseteq_c \text{NP}_{\{k-1\}} V$ then $\text{PH} = \text{NP}^{\text{NP}}$.*

Theorem 9. *Let $k \geq 2$ and d , $1 \leq d \leq k - 1$, be integers. Let $A, B \subseteq \mathbb{N}^+$ be such that $\binom{k-1}{k-d} \in A$, $\binom{k}{k-d} \in A$, and $\max\{i \mid i \in B \text{ and } i \leq \binom{k}{k-d}\} \leq \lceil \frac{k}{d} \rceil - 1$. Then $\text{NP}_A V \subseteq_c \text{NP}_B V$ implies $\text{PH} = \text{NP}^{\text{NP}}$.*

In fact, Theorem 9, a necessary-condition theorem quite different from Theorem 8, has some very useful corollaries. For example, the necessary condition of Naik et al. (Theorem 4) follows immediately from Theorem 9 by plugging in $d = 1$ into the above; in fact, doing so gives a statement:

($\star\star$) For each $k \geq 1$, if $\text{NP}_{\{1, k+1\}} V \subseteq_c \text{NP}_{\{1, \dots, k\}} V$ then $\text{PH} = \text{NP}^{\text{NP}}$.

that is even stronger than the Naik et al. result. However, we note that if one closely examines the proof of Naik et al. one can in fact see that their *proof* establishes ($\star\star$).

Theorem 9 yields other interesting necessary conditions. As an example, from the $d = 2$ case we can certainly conclude the following result.

Corollary 2. *For each $k > 2$, if $\text{NP}_{\{k-1, \binom{k}{2}\}} V \subseteq_c \text{NP}_{\{1, \dots, \lceil k/2 \rceil - 1\}} V$ then $\text{PH} = \text{NP}^{\text{NP}}$.*

4 Lowness Results

We now prove another strengthening of the result of Naik et al. [15] stated here as Theorem 4. Namely, we show a lowness result—a general result about the simpleness of sets having certain properties—from which Naik et al.’s Theorem 4 is a consequence. Informally, lowness captures the level of the polynomial hierarchy, if any, at which a given NP set becomes worthless as an oracle—the level at which it gives that level no more additional information than would the empty set. Of interest to us will be the class of sets for which this level is two.

Definition 1. [15] *For any integer $k > 0$ and any function class $\mathcal{FC}$ we say that a set A is $\mathcal{FC}$ - k -selective if there is a function $f \in \mathcal{FC}$ such that for every k distinct strings $b_1, \dots, b_k$,*

1. *every output of $f(b_1, \dots, b_k)$ is a cardinality $k - 1$ subset of $\{b_1, \dots, b_k\}$ and*
2. *if $\|\{b_1, \dots, b_k\} \cap A\| \geq k - 1$, then $f(b_1, \dots, b_k)$ has at least one output and each set output by $f(b_1, \dots, b_k)$ is a subset of A .*

Definition 2. [18] $\text{Low}_2 = \{A \in \text{NP} \mid \text{NP}^{\text{NP}^A} = \text{NP}^{\text{NP}}\}$.

Theorem 10. *For each $k \in \{2, 3, \dots\}$, it holds that every $\text{NP}_{\{1, \dots, k-1\}} V$ - k -selective NP set belongs to Low_2 .*

We will postpone proving this theorem until Section 5.

Theorem 4 certainly follows from Theorem 10. In fact, recall the stronger form of Theorem 4 that we noted in Section 3 (marked “($\star\star$)”). Since SAT is $\text{NP}_{\{1, k\}} V$ - k -selective [15], even this stronger form of Theorem 4 follows immediately from Theorem 10. More generally, Theorem 10 establishes the simpleness of NP’s $\text{NP}_{\{1, \dots, k-1\}} V$ - k -selective sets.

5 A Unified Strengthening

Note that in the previous sections we have stated extensions of the work of Naik et al. (Theorem 4 in two incomparable ways, namely providing as Theorem 9 a broader necessary condition and as Theorem 10 a general lowness theorem that implied the Naik et al. result. It is very natural to ask whether our two results can be unified, via proving a lowness result that itself implies not just Theorem 4 but all the necessary conditions we identify in this paper. In fact, the answer is yes. We have the following result, which provides exactly such a unification.

Definition 3. Let $k \geq 2$ be an integer. A parameter tuple for input size k is a $k + 3$ tuple $\Lambda = \langle \ell_0, \dots, \ell_{k-1}, \alpha, \beta, \gamma \rangle$ of nonnegative integers such that

- at least one of $\ell_1, \dots, \ell_{k-1}$ is positive,
- $0 \leq \alpha \leq \sum_{i=1}^{k-1} \binom{k-1}{i} \ell_i$,
- $0 \leq \beta \leq \sum_{i=1}^{k-1} ((\binom{k}{i}) - \binom{k-1}{i}) \ell_i$, and
- $0 \leq \gamma \leq \ell_0$.

Definition 4. Let $k \geq 2$ be an integer. Let Λ be a parameter tuple for input size k . Let $\mathcal{FC}$ be a class of multivalued functions. A language A is $\mathcal{FC}$ -(k, Λ)-selective if there is some $f \in \mathcal{FC}$ such that, for every set X of k distinct strings $x_1, \dots, x_k$, the following properties hold:

1. Each output value of $f(X)$ belongs to the union of the following three classes of strings:
 - Class A** $\{\langle i, j, W \rangle \mid 1 \leq i \leq k-1 \text{ and } 1 \leq j \leq \ell_i \text{ and } W \text{ is a cardinality } i \text{ subset of } X \cap A\}$;
 - Class B** $\{\langle i, j, W \rangle \mid 1 \leq i \leq k-1 \text{ and } 1 \leq j \leq \ell_i \text{ and } W \text{ is a cardinality } i \text{ subset of } X \text{ containing at least one member of } \overline{A}\}$; and
 - Class C** $\{\langle 0, j \rangle \mid 0 \leq j \leq \ell_0\}$;
2. If $\|X \cap A\| = k-1$, then $f(X)$ should output no more than α Class A strings, no more than β Class B strings, and no more than γ Class C strings.

Definition 5. Let $k \geq 2$ and $\Lambda = \langle \ell_0, \dots, \ell_{k-1}, \alpha, \beta, \gamma \rangle$ be a parameter tuple for input size k . Let $B \subseteq \mathbb{N}^+$ be nonempty and finite. Define the predicate

$$Q(k, \Lambda, B) = [\alpha + \beta + \gamma \geq \min\{i \mid i \in B\} \text{ and } k\Delta < \sum_{i=1}^{k-1} (k-i)t_i],$$

where Δ and $t_1, \dots, t_{k-1}$ are defined as follows:

- $s_k = 0$, $s'_k = 0$, and for each d , $1 \leq d \leq k-1$, $s_d = \sum_{i=d}^{k-1} \binom{k-1}{i} \ell_i$ and $s'_d = \sum_{i=d}^{k-1} \binom{k}{i} \ell_i$.
- $\Delta = (s_1 + \beta + \gamma) - \min\{i \mid i \in B\}$.
- $\Delta' = \max\{0, s'_1 - \max\{i \in B \mid i \leq s'_1 + \ell_0\}\}$.
- For each d , $1 \leq d \leq k-1$, $t_d = \max\{0, \min\{\Delta' - s'_{d+1}, s'_d - s'_{d+1}\}\}$.

Note that in the above definition the $\max\{i \in B \mid i \leq s'_1 + \ell_0\}$, which is used to define the second conjunct of Q , is well-defined conditionally upon the first conjunct holding, as in that case we have $\min\{i \mid i \in B\} \leq \beta + \gamma \leq (\sum_{i=1}^{k-1} \binom{k}{i} - \binom{k-1}{i} \ell_i) + \ell_0 \leq s'_1 + \ell_0$.

Theorem 11. *Let $k \geq 2$ be an integer, let Λ be a parameter tuple for input size k , and let B be a nonempty finite set of positive integers such that $Q(k, \Lambda, B)$ holds. Then every $\text{NP}_B\text{V}-(k, \Lambda)$ -selective set in NP belongs to Low_2 .*

Proposition 2. *Let $k \geq 2$ be an integer and $\Lambda = \langle \ell_0, \dots, \ell_{k-1}, \alpha, \beta, \gamma \rangle$ be a parameter tuple for input size k . Let $H \subset \mathbb{N}^+$ be any finite set containing $\alpha + \beta + \gamma$ and let $\sigma = \sum_{i=0}^{k-1} \binom{k}{i} \ell_i$. Then every language $A \in \text{NP}$ is $\text{NP}_H\text{V}-(k, \Lambda)$ -selective.*

Due to the page limit we omit here the proofs of Theorem 11 and Proposition 2. An interested reader may refer to the full paper [10].

From Theorem 11 and Proposition 2, we have the following Corollary.

Corollary 3. *Let $k \geq 2$ be an integer and $\Lambda = \langle \ell_0, \dots, \ell_{k-1}, \alpha, \beta, \gamma \rangle$ a parameter tuple for input size k . Let $H \subset \mathbb{N}^+$ be any finite set containing $\alpha + \beta + \gamma$ and $\sigma = \sum_{i=0}^{k-1} \binom{k}{i} \ell_i$. Let B be a finite set of positive integers such that $Q(k, \Lambda, B)$ holds and such that $\min\{i \mid i \in B\} \leq \min\{i \mid i \in H\}$. Then $\text{NP}_H\text{V} \subseteq_c \text{NP}_B\text{V}$ implies $\text{PH} = \Sigma_2^P$.*

Now we turn to proving Theorems 8, 9, and 10 using the unified lowness theorem (Theorem 11).

Proof of Theorem 8. Let A, B, c, d, e , and δ be as in the hypothesis of the theorem. Let $k = 2$ and define a parameter tuple for size 2, $\Lambda = \langle \ell_0, \ell_1, \alpha, \beta, \gamma \rangle$ by:

- $\ell_0 = e, \ell_1 = d$,
- $\alpha = d, \beta = c - d$, and $\gamma = 0$.

Let L be an arbitrary language in NP. By Proposition 2, L is $\text{NP}_A\text{V}-(2, \Lambda)$ -selective. Now suppose $\text{NP}_A\text{V} \subseteq_c \text{NP}_B\text{V}$. Then $s_1 = \alpha$ and $c = \alpha + \beta + \gamma$. Since $c - \delta \geq \min\{i \mid i \in B\}$, $\Delta \leq \delta$. Also, $2d + e = s'_1 + \gamma$. Thus, $2d - (2\delta + 1) \geq \max\{i \in B \mid i \leq 2d + e\}$ implies $\Delta' \geq 2\delta + 1$. Thus, $t_1 = \Delta'$. So, we have $\sum_{i=1}^k t_i(k-i) = t_1 \cdot 1 = t_1 = \Delta' > 2\delta \geq 2\Delta$, and so, $Q(2, \Lambda, B)$ holds. Hence, L is Low_2 . Since L is an arbitrary NP language, this implies $\text{PH} = \text{NP}^{\text{NP}}$.

Proof of Theorem 9. Let k, d, A , and B be as in the hypothesis of the theorem. By Proposition 1, we can assume that $\min\{i \mid i \in B\} \leq \binom{k-1}{k-d}$; otherwise, the statement of the Theorem is *false* $\Rightarrow \text{PH} = \text{NP}^{\text{NP}}$. Define a parameter tuple for size k , $\Lambda = \langle \ell_0, \dots, \ell_{k-1}, \alpha, \beta, \gamma \rangle$ by:

- $\ell_0 = \dots = \ell_{k-d-1} = \ell_{k-d+1} = \dots = \ell_{k-1} = 0, \ell_{k-d} = 1$,
- $\alpha = \binom{k-1}{k-d}$, and $\beta = \gamma = 0$.

Let L be an arbitrary language in NP. By Proposition 2, L is NP_AV -(k, Λ)-selective. Now $\text{NP}_A\text{V} \subseteq_c \text{NP}_B\text{V}$ implies $\Delta \leq \binom{k-1}{k-d} - 1$ and $\Delta' \geq \binom{k}{k-d} - \lceil \frac{k}{d} \rceil + 1$. Since $\Delta' < \binom{k}{k-d}$ and $\ell_i \neq 0$ holds only for $i = k-d$, we have that $t_{k-d} = \Delta'$ and that, for every i , $0 \leq i \leq k-1$, such that $i \neq k-d$, $t_i = 0$. So, $\sum_{i=0}^{k-1} t_i(k-i) = k\Delta' \geq d(\binom{k}{k-d} - \lceil \frac{k}{d} \rceil + 1) > d(\binom{k}{k-d} - \frac{k}{d}) = k(\binom{k-1}{k-d} - 1) = k\Delta$. This implies that $Q(k, \Lambda, B)$ holds, so by Theorem 11 L is in Low_2 . Since L is an arbitrary NP set, this implies $\text{PH} = \text{NP}^{\text{NP}}$.

Proof of Theorem 10. Let $k \geq 2$. Define a parameter tuple for size k , $\Lambda = \langle \ell_0, \dots, \ell_{k-1}, \alpha, \beta, \gamma \rangle$ by:

- $\ell_0 = \dots = \ell_{k-2} = 0$, $\ell_{k-1} = 1$,
- $\alpha = \binom{k-1}{k-1} = 1$, and $\beta = \gamma = 0$.

Let $B = \{1, \dots, k-1\}$. The family of all NP_BV - k -selective sets is precisely that of all NP_BV -(k, Λ)-selective sets. Since $\alpha + \beta + \gamma = 1$, Δ is necessarily 0. On the other hand, the hypothesis of the theorem implies $\Delta' > 0$. So, Q holds. The rest of the proof is the same.

6 Conclusion

In this paper we gave a condition that we conjecture is, assuming that the polynomial hierarchy does not collapse, necessary and sufficient for determining for finite cardinality types A and B whether $\text{NP}_A\text{V} \subseteq_c \text{NP}_B\text{V}$, i.e., informally, for determining the ways in which solution cardinality can be pruned. We proved our condition to be (unconditionally) sufficient. We also established a necessary condition assuming that the polynomial hierarchy does not collapse. However, our necessary-condition theorem is not yet strong enough to match our sufficient condition.

Nonetheless, we hope that in time this will be established and we recommend as an interesting open problem the issue of proving the Narrowing-Gap Conjecture. Certainly, there are many similar cases in the literature where similar complete characterizations have been completed or interestingly posed. Hemaspaandra, Hempel, and Wechsung ([8], see also, e.g., [22] and the survey [7]) have under the assumption that the polynomial hierarchy does not collapse, completely characterized (for pairs of levels of the boolean hierarchy) when query order matters. Kosub and Wagner [14] have posed, and made powerful progress towards, a complete characterization regarding their boolean hierarchy of NP-partitions. Also, Kosub [13] proves that for each cardinality-type pair A and B violating the Narrowing-Gap Conjecture, there is an oracle $W(A, B)$ such that $\text{NP}_A\text{V}^{W(A, B)} \not\subseteq_c \text{NP}_B\text{V}^{W(A, B)}$. Finally, we commend to the reader the work of Durand, Hermann, and Kolaitis [5], which defines and studies “subtractive reductions.”

References

1. R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13(3):461–487, 1984.
2. R. Book, T. Long, and A. Selman. Qualitative relativizations of complexity classes. *Journal of Computer and System Sciences*, 30(3):395–413, 1985.
3. D. Bovet, P. Crescenzi, and R. Silvestri. A uniform approach to define complexity classes. *Theoretical Computer Science*, 104(2):263–283, 1992.
4. J. Cai, T. Gundermann, J. Hartmanis, L. Hemachandra, V. Sewelson, K. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.
5. A. Durand, M. Hermann, and P. Kolaitis. Subtractive reductions and complete problems for counting complexity classes. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*. Springer-Verlag *Lecture Notes in Computer Science*, August/September 2000. To appear.
6. S. Fenner, L. Fortnow, A. Naik, and J. Rogers. On inverting onto functions. In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, pages 213–222. IEEE Computer Society Press, May 1996.
7. E. Hemaspaandra, L. Hemaspaandra, and H. Hempel. An introduction to query order. *Bulletin of the EATCS*, 63:93–107, 1997.
8. L. Hemaspaandra, H. Hempel, and G. Wechsung. Query order. *SIAM Journal on Computing*, 28(2):637–651, 1999.
9. L. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, 25(4):697–708, 1996.
10. L. Hemaspaandra, M. Ogihara, and G. Wechsung. On reducing the number of solutions of NP functions. Technical Report TR-727, Department of Computer Science, University of Rochester, Rochester, NY, January 2000. Revised, March 2000.
11. L. Hemaspaandra, J. Rothe, and G. Wechsung. Easy sets and hard certificate schemes. *Acta Informatica*, 34(11):859–879, 1997.
12. B. Jenner and J. Torán. The complexity of obtaining solutions for problems in NP. In L. Hemaspaandra and A. Selman, editors, *Complexity Theory Retrospective II*. Springer-Verlag, 1997.
13. S. Kosub. On NP-partitions over posets with an application to reducing the set of solutions of NP problems. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science*. Springer-Verlag *Lecture Notes in Computer Science*, August/September 2000. To appear.
14. S. Kosub and K. Wagner. The boolean hierarchy of NP-partitions. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, pages 157–168. Springer-Verlag *Lecture Notes in Computer Science* #1770, February 2000.
15. A. Naik, J. Rogers, J. Royer, and A. Selman. A hierarchy based on output multiplicity. *Theoretical Computer Science*, 207(1):131–157, 1998.
16. M. Ogihara. Functions computable with limited access to NP. *Information Processing Letters*, 58:35–38, 1996.
17. M. Ogiwara and L. Hemachandra. A complexity theory for feasible closure properties. *Journal of Computer and System Sciences*, 46(3):295–325, 1993.
18. U. Schöning. A low and a high hierarchy within NP. *Journal of Computer and System Sciences*, 27:14–28, 1983.

19. A. Selman. A taxonomy of complexity classes of functions. *Journal of Computer and System Sciences*, 48(2):357–381, 1994.
20. A. Selman. Much ado about functions. In *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, pages 198–212. IEEE Computer Society Press, May 1996.
21. N. Vereshchagin. Relativizable and nonrelativizable theorems in the polynomial theory of algorithms. *Russian Academy of Sciences–Izvestiya–Mathematics*, 42(2):261–298, 1994.
22. K. Wagner. A note on parallel queries and the symmetric-difference hierarchy. *Information Processing Letters*, 66(1):13–20, 1998.

Regular Collections of Message Sequence Charts (Extended Abstract)

Jesper G. Henriksen¹, Madhavan Mukund^{2,*},
K. Narayan Kumar^{2,*}, and P.S. Thiagarajan²

¹ BRICS^{**}, University of Aarhus, Denmark. Email: gulmann@brics.dk

² Chennai Mathematical Institute, Chennai, India
Email: {madhavan,kumar,pst}@smi.ernet.in

Abstract. Message Sequence Charts (MSCs) are an attractive visual formalism used during the early stages of design in domains such as telecommunication software. A popular mechanism for generating a collection of MSCs is a Hierarchical Message Sequence Chart (HMSC). However, not all HMSCs describe collections of MSCs that can be “realized” as a finite-state device. Our main goal is to pin down this notion of realizability. We propose an independent notion of *regularity* for collections of MSCs and explore its basic properties. In particular, we characterize regular collections of MSCs in terms of finite-state distributed automata called bounded message-passing automata, in which a set of sequential processes communicate with each other asynchronously over bounded FIFO channels. We also provide a logical characterization in terms of a natural monadic second-order logic interpreted over MSCs. It turns out that realizable collections of MSCs as specified by HMSCs constitute a strict subclass of the regular collections of MSCs.

1 Introduction

Message sequence charts (MSCs) are an appealing visual formalism often used to capture system requirements in the early stages of design. They are particularly suited for describing scenarios for distributed telecommunication software [12, 19]. They have also been called timing sequence diagrams, message flow diagrams and object interaction diagrams and are used in a number of software engineering methodologies [4,9,19]. In its basic form, an MSC depicts the exchange of messages between the processes of a distributed system along a single partially-ordered execution. A collection of MSCs is used to capture the scenarios that a designer might want the system to exhibit (or avoid).

Given the requirements in the form of a collection of MSCs, one can hope to do formal analysis and discover design errors at an early stage. A natural question in this context is to identify when a collection of MSCs is amenable to formal analysis. A related issue is how to represent such collections.

* Supported in part by IFCPAR Project 2102-1.

** Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

A standard way to generate a collection of MSCs is to use a Hierarchical Message Sequence Chart (HMSC) [15]. An HMSC is a finite directed graph in which each node is labelled, in turn, by an HMSC. The labels on the nodes are not permitted to refer to each other. From an HMSC we can derive an equivalent Message Sequence Graph (MSG) [1] by flattening out the hierarchical labelling to obtain a graph where each node is labelled by a simple MSC. An MSG defines a collection of MSCs obtained by concatenating the MSCs labelling each path from an initial vertex to a terminal vertex. Though HMSCs provide more succinct specifications than MSGs, they are only as expressive as MSGs. Thus, one often restricts one's attention to characterizing structural properties of MSGs rather than of HMSCs [2,17,18].

In [2], it is shown that *bounded* MSGs define reasonable collections of MSCs—the collection of MSCs generated by a bounded MSG can be represented as a regular string language. Thus, behaviours captured by bounded MSGs can, in principle, be realized as finite-state automata. In general, the collection of MSCs defined by an *arbitrary* MSG is not realizable in this sense. A characterization of the collections of MSCs definable using bounded MSGs is provided in [11].

The main goal of this paper is to pin down this notion of realizability in terms of a notion of *regularity* for collections of MSCs. One consequence of our study is that our definition of regularity provides a general and robust setting for studying collections of MSCs. A second consequence, which follows from the results in [11], is that bounded MSGs define a strict subclass of regular collections of MSCs. A final consequence is that our notion addresses an important issue raised in [6]; namely, how to convert requirements as specified by MSCs into distributed, state-based specifications.

Another motivation for focussing on regularity is that this notion has turned out to be very fruitful in a variety of contexts including finite (and infinite) strings, trees and restricted partial orders known as Mazurkiewicz traces [7,21]. In all these settings there is a representation of regular collections in terms of finite-state devices. There is also an accompanying monadic second-order logic that usually induces temporal logics using which one can reason about such collections [21]. One can then develop automated model-checking procedures for verifying properties specified in these temporal logics. In this context, the associated finite-state devices representing the regular collections often play a very useful role [22].

We show here that our notion of regular MSC languages fits in nicely with a related notion of a finite-state device, as also a monadic second-order logic. We fix a finite set of processes $\mathcal{P}$ and consider $\mathcal{M}$, the universe of MSCs defined over the set $\mathcal{P}$. An MSC in $\mathcal{M}$ can be viewed as a partial order labelled using a finite alphabet Σ that is canonically fixed by $\mathcal{P}$. We say that $L \subseteq \mathcal{M}$ is regular if the set of all linearizations of all members of L constitutes a regular subset of Σ^* . A crucial point is that the universe $\mathcal{M}$ is itself *not* regular according to our definition, unlike the classical setting of strings (or trees or Mazurkiewicz traces). This fact has a strong bearing on the automata-theoretic and logical formulations in our work.

It turns out that regular MSC languages can be stratified using the concept of *bounds*. An MSC is said to be B -bounded for a natural number B if at every “prefix” of the MSC and for every pair of processes (p, q) there are at most B messages that p has sent to q that have yet to be received by q . An MSC language is B -bounded if every member of the language is B -bounded. Fortunately, for every regular MSC language L we can effectively compute a (minimal) bound B such that L is B -bounded. This leads to our automaton model called B -bounded message-passing automata. The components of such an automaton correspond to the processes in $\mathcal{P}$. These components communicate with each other over (potentially unbounded) FIFO channels. We say that a message-passing automaton is B -bounded if, during its operation, it is never the case that a channel contains more than B messages. We establish a precise correspondence between B -bounded message-passing automata and B -bounded regular MSC languages. In a similar vein, we formulate a natural monadic second-order logic $\text{MSO}(\mathcal{P}, B)$ interpreted over B -bounded MSCs. We then show that B -bounded regular MSC languages are exactly those that are definable in $\text{MSO}(\mathcal{P}, B)$.

In related work, a number of studies are available that are concerned with individual MSCs in terms of their semantics and properties [1,13]. As pointed out earlier, a nice way to generate a collection of MSCs is to use an MSG. A variety of algorithms have been developed for MSGs in the literature—for instance, pattern matching [14,17,18] and detection of process divergence and non-local choice [3]. A systematic account of the various model-checking problems associated with MSGs and their complexities is given in [2].

In this paper, we confine our attention to *finite* MSCs. The issues investigated here have, at present, no counterparts in the infinite setting. We feel, however, that our results will serve as a launching pad for a similar account concerning infinite MSCs. This should then lead to the design of appropriate temporal logics and automata-theoretic solutions (based on message-passing automata) to model-checking problems for these logics.

The paper is organized as follows. In the next section we introduce MSCs and regular MSC languages. In Section 3 we establish our automata-theoretic characterization and, in Section 4, the logical characterization. While doing so, we borrow one basic result and a couple of proof techniques from the theory of Mazurkiewicz traces [7]. However, we need to modify some of these techniques in a non-trivial way (especially in the setting of automata) due to the asymmetric flow of information via messages in the MSC setting, as opposed to the symmetric information flow via handshake communication in the trace setting. Due to lack of space, we provide only proof ideas. Detailed proofs are available in [10].

2 Regular MSC Languages

Through the rest of the paper, 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 .

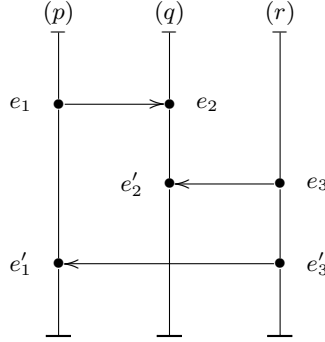


Fig. 1. An example MSC over $\{p, q, r\}$.

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 $\lambda : E \rightarrow \Sigma$ 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 relation $R_c = \{(e, e') \mid \lambda(e) = p!q, \lambda(e') = q?p \text{ and } \downarrow e \cap E_{p!q} = \downarrow e' \cap E_{q?p}\}$. Finally, for each $p \in \mathcal{P}$, we define the relation $R_p = (E_p \times E_p) \cap \leq$.

An MSC (over $\mathcal{P}$) is a *finite* Σ -labelled poset $M = (E, \leq, \lambda)$ that 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 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 smaller (relative to $\leq$) events appearing above the larger 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.

In the literature [1,18] one sometimes considers a more generous notion of linearization where two *adjacent* receive actions in a process corresponding to

messages from *different* senders are deemed to be causally independent. For instance, $p!q \ r!q \ q?p \ q?p \ r!p \ p?p$ would also be a valid linearization of the MSC in Figure 1. All our results go through with suitable modifications even in the presence of this more generous notion of linearization.

Henceforth, we will 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.

To directly characterize the subsets of Σ^* that correspond to regular MSC languages, we proceed as follows. Let $Com = \{(p!q, q?p) \mid (p, q) \in Ch\}$. For $\tau \in \Sigma^*$ and $a \in \Sigma$, let $|\tau|_a$ denote the number of times a appears in τ . We say that $\sigma \in \Sigma^*$ is *proper* if for every prefix τ of σ and every pair $(a, b) \in Com$, $|\tau|_a \geq |\tau|_b$. We say that σ is *complete* if σ is proper and $|\sigma|_a = |\sigma|_b$ for every $(a, b) \in Com$. Next we define a *context-sensitive* independence relation $I \subseteq \Sigma^* \times (\Sigma \times \Sigma)$ as follows: $(\sigma, a, b) \in I$ if σab is proper, $a \in \Sigma_p$ and $b \in \Sigma_q$ for distinct processes p and q , and if $(a, b) \in Com$ then $|\sigma|_a > |\sigma|_b$. Observe that if $(\sigma, a, b) \in I$ then $(\sigma, b, a) \in I$.

Let $\Sigma^\circ = \{\sigma \mid \sigma \in \Sigma^* \text{ and } \sigma \text{ is complete}\}$. We then define $\sim \subseteq \Sigma^\circ \times \Sigma^\circ$ to be the least equivalence relation such that if $\sigma = \sigma_1 ab \sigma_2$, $\sigma' = \sigma_1 ba \sigma_2$ and $(\sigma_1, a, b) \in I$ then $\sigma \sim \sigma'$. It is important to note that $\sim$ is defined over Σ° (and not Σ^*). It is easy to verify that for each $M \in \mathcal{M}_{\mathcal{P}}$, $Lin(M)$ is a subset of Σ° and is in fact a $\sim$ -equivalence class over Σ° .

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 $L = \bigcup \{Lin(M) \mid M \in \mathcal{L}\}$. It is easy to see that $L \subseteq \Sigma^*$ is a regular string MSC language if and only if L is a regular subset of Σ^* , every word in L is complete and L is $\sim$ -closed (that is, for each $\sigma \in L$, if $\sigma \in L$ and $\sigma \sim \sigma'$ then $\sigma' \in L$). Clearly 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 channel-capacity function $\mathcal{K}_s : Ch \rightarrow \mathbb{N}$ that 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 $\delta(s'_1, a) = s_2$.

These conditions can be checked in time linear in the size of δ . 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 complete word 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 $L \subseteq \Sigma^\circ$ is B -bounded if every word $\sigma \in L$ is B -bounded. Let L be a regular MSC language and let $\mathcal{A} = (S, \Sigma, s_{in}, \delta, F)$ be its minimal DFA, as described above, with capacity functions $\{\mathcal{K}_s\}_{s \in S}$. Let $B_L = \max_{s \in S, c \in Ch} \mathcal{K}_s(c)$. Then it is easy to see that L is B_L -bounded and that B_L can be effectively computed from $\mathcal{A}$. Finally, we shall say that the MSC M is B -bounded if every string in $Lin(M)$ is B -bounded. A collection of MSCs is B -bounded if every member of the collection is B -bounded.

3 An Automata-Theoretic Characterization

Recall that the set of processes $\mathcal{P}$ determines the communication alphabet Σ and that for $p \in \mathcal{P}$, Σ_p denotes the actions in which process p participates.

Definition 3.1. A message-passing automaton over Σ is a structure $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in \mathcal{P}}, \Delta, s_{in}, \mathcal{F})$ where

- Δ is a finite alphabet of messages.
- Each component $\mathcal{A}_p$ is of the form $(S_p, \rightarrow_p)$ where
 - S_p is a finite set of p -local states.
 - $\rightarrow_p \subseteq S_p \times \Sigma_p \times \Delta \times S_p$ is the p -local transition relation.
- $s_{in} \in \prod_{p \in \mathcal{P}} S_p$ is the global initial state.
- $\mathcal{F} \subseteq \prod_{p \in \mathcal{P}} S_p$ is the set of global final states.

The local transition relation $\rightarrow_p$ specifies how process p sends and receives messages. The transition $(s, p!q, m, s')$ specifies that when p is in the state s , it can send the message m to q by executing the action $p!q$ and move to the state s' . The message m is, as a result, appended to the queue in channel (p, q) . Similarly, the transition $(s, p?q, m, s')$ signifies that in the state s , the process p can receive the message m from q by executing the action $p?q$ and move to the state s' . The message m is removed from the head of the queue in channel (q, p) .

The set of global states of $\mathcal{A}$ is given by $\prod_{p \in \mathcal{P}} S_p$. For a global state s , we let s_p denote the p th component of s . A *configuration* is a pair (s, χ) where s is a global state and $\chi : Ch \rightarrow \Delta^*$ is the *channel state* that specifies the queue of messages currently residing in each channel c . The *initial configuration* of $\mathcal{A}$ is $(s_{in}, \chi_\varepsilon)$ where $\chi_\varepsilon(c)$ is the empty string ε for every channel c . The set of *final configurations* of $\mathcal{A}$ is $\mathcal{F} \times \{\chi_\varepsilon\}$.

We now define the set of reachable configurations $Conf_{\mathcal{A}}$ and the global transition relation $\Rightarrow \subseteq Conf_{\mathcal{A}} \times \Sigma \times Conf_{\mathcal{A}}$ inductively as follows:

- $(s_{in}, \chi_\varepsilon) \in Conf_{\mathcal{A}}$.
- Suppose $(s, \chi) \in Conf_{\mathcal{A}}$, (s', χ') is a configuration and $(s_p, p!q, m, s'_p) \in \rightarrow_p$ such that the following conditions are satisfied:

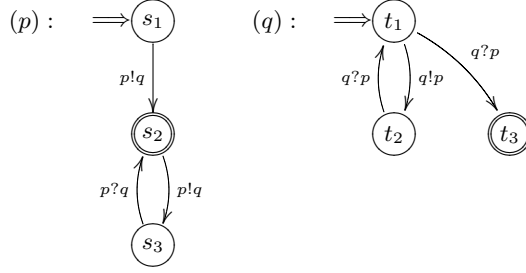


Fig. 2. A 3-bounded message-passing automaton.

- $r \neq p$ implies $s_r = s'_r$ for each $r \in \mathcal{P}$.
- $\chi'((p, q)) = \chi((p, q)) \cdot m$ and for $c \neq (p, q)$, $\chi'(c) = \chi(c)$.

Then $(s, \chi) \xrightarrow{p!q} (s', \chi')$ and $(s', \chi') \in \text{Conf}_{\mathcal{A}}$.

- Suppose $(s, \chi) \in \text{Conf}_{\mathcal{A}}$, (s', χ') is a configuration and $(s_p, p?q, m, s'_p) \in \rightarrow_p$ such that the following conditions are satisfied:

- $r \neq p$ implies $s_r = s'_r$ for each $r \in \mathcal{P}$.
- $\chi((q, p)) = m \cdot \chi'((q, p))$ and for $c \neq (q, p)$, $\chi'(c) = \chi(c)$.

Then $(s, \chi) \xrightarrow{p?q} (s', \chi')$ and $(s', \chi') \in \text{Conf}_{\mathcal{A}}$.

Let $\sigma \in \Sigma^*$. A run of $\mathcal{A}$ over σ is a map $\rho : \text{Pre}(\sigma) \rightarrow \text{Conf}_{\mathcal{A}}$ (where $\text{Pre}(\sigma)$ is the set of prefixes of σ) such that $\rho(\varepsilon) = (s_{\text{in}}, \chi_\varepsilon)$ and for each $\tau a \in \text{Pre}(\sigma)$, $\rho(\tau) \xrightarrow{a} \rho(\tau a)$. The run ρ is *accepting* if $\rho(\sigma)$ is a final configuration. We define $L(\mathcal{A}) = \{\sigma \mid \mathcal{A} \text{ has an accepting run over } \sigma\}$. It is easy to see that every member of $L(\mathcal{A})$ is complete and $L(\mathcal{A})$ is $\sim$ -closed.

Clearly, $L(\mathcal{A})$ need not be regular. Consider, for instance, a message-passing automaton for the canonical producer-consumer system in which the producer p sends an arbitrary number of messages to the consumer q . Since we can reorder all the $p!q$ actions to be performed before all the $q?p$ actions, the queue in channel (p, q) can grow arbitrarily long. Hence, the reachable configurations of this system are not bounded and the corresponding language is not regular.

For $B \in \mathbb{N}$, we say that a configuration (s, χ) of the message-passing automaton $\mathcal{A}$ is *B-bounded* if for every channel $c \in \text{Ch}$, it is the case that $|\chi(c)| \leq B$. We say that $\mathcal{A}$ is a *B-bounded automaton* if every reachable configuration $(s, \chi) \in \text{Conf}_{\mathcal{A}}$ is *B-bounded*. It is not difficult to show that given a message-passing automaton $\mathcal{A}$ and a bound $B \in \mathbb{N}$, one can decide whether $\mathcal{A}$ is *B-bounded*. Figure 2 shows an example of a 3-bounded message-passing automaton with two components, p and q . In this example, the message alphabet is a singleton, and is hence omitted. The initial state is (s_1, t_1) and there is only one final state, (s_2, t_3) . This automaton accepts an infinite set of MSCs, none of which can be expressed as the concatenation of two or more non-trivial MSCs. It follows easily that the MSC language accepted by this automaton cannot be represented by an MSG.

Lemma 3.2. *Let $\mathcal{A}$ be a B -bounded automaton over Σ . Then $L(\mathcal{A})$ is a B -bounded regular MSC language.*

This result follows from the definitions and it constitutes the easy half of the characterization we wish to obtain. The difficult half is :

Lemma 3.3. *Let $L \subseteq \Sigma^*$ be a B -bounded regular MSC language. Then there exists a B -bounded message-passing automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L$.*

The first step in the proof is to view L as a regular Mazurkiewicz trace language over an appropriate trace alphabet and apply Zielonka's theorem [23] to obtain an asynchronous automaton recognizing L . The second step, which is the hard part, is to simulate this asynchronous automaton by a B -bounded message passing automaton. This simulation makes crucial use of the technique developed in [16] for locally maintaining the latest information about other processes in a message-passing system.

We say that $\mathcal{A}$ is a bounded message-passing automaton if $\mathcal{A}$ is B -bounded for some $B \in \mathbb{N}$. The main result of this section is an easy consequence of the two previous lemmas.

Theorem 3.4. *Let $L \subseteq \Sigma^*$ be a regular MSC language. Then there exists a bounded message-passing automaton $\mathcal{A}$ over Σ such that $L(\mathcal{A}) = L$.*

4 A Logical Characterization

We formulate a monadic second-order logic that characterizes regular B -bounded MSC languages for each fixed $B \in \mathbb{N}$. Thus our logic will be parameterized by a pair $(\mathcal{P}, B)$. For convenience, we fix $B \in \mathbb{N}$ through the rest of the section. As usual, we shall assume a supply of individual variables $x, y, \dots$, a supply of set variables $X, Y, \dots$, and a family of unary predicate symbols $\{Q_a\}_{a \in \Sigma}$. The syntax of the logic is then given by:

$$\text{MSO}(\mathcal{P}, B) ::= Q_a(x) \mid x \in X \mid x \leq y \mid \neg\varphi \mid \varphi \vee \varphi \mid (\exists x)\varphi \mid (\exists X)\varphi$$

Thus the syntax does not reflect any information about B or the structural features of an MSC. These aspects will be dealt with in the semantics. Let $\mathcal{M}_{\mathcal{P}, B}$ be the set of B -bounded MSCs over $\mathcal{P}$. The formulas of our logic are interpreted over the members of $\mathcal{M}_{\mathcal{P}, B}$. Let $M = (E, \leq, \lambda)$ be an MSC in $\mathcal{M}_{\mathcal{P}, B}$ and $\mathcal{I}$ be an interpretation that assigns to each individual variable a member $\mathcal{I}(x)$ in E and to each set variable X a subset $\mathcal{I}(X)$ of E . Then $M \models_{\mathcal{I}} \varphi$ denotes that M satisfies φ under $\mathcal{I}$. This notion is defined in the expected manner. For instance, $M \models_{\mathcal{I}} Q_a(x)$ if $\lambda(\mathcal{I}(x)) = a$, $M \models_{\mathcal{I}} x \leq y$ if $\mathcal{I}(x) \leq \mathcal{I}(y)$ etc. For convenience, we have used $\leq$ to denote both the predicate symbol in the logic and the corresponding causality relation in the model M .

As usual, φ is a sentence if there are no free occurrences of individual or set variables in φ . With each sentence φ we can associate an MSC language $\mathcal{L}_{\varphi} = \{M \in \mathcal{M}_{\mathcal{P}, B} \mid M \models \varphi\}$. We say that $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P}, B}$ is $\text{MSO}(\mathcal{P}, B)$ -definable

if there exists a sentence φ such that $\mathcal{L}_\varphi = \mathcal{L}$. We wish to argue that $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P},B}$ is $\text{MSO}(\mathcal{P}, B)$ -definable if and only if it is a B -bounded regular MSC language. It turns out the techniques used for proving a similar result in the theory of traces [8] can be suitably modified to derive our result.

Lemma 4.1. *Let φ be a sentence in $\text{MSO}(\mathcal{P}, B)$. Then $\mathcal{L}_\varphi$ is a B -bounded regular MSC language.*

Proof Sketch: The fact that $\mathcal{L}_\varphi$ is B -bounded follows from the semantics and hence we just need to establish regularity. Consider $\text{MSO}(\Sigma)$, the monadic second-order theory of finite strings in Σ^* . This logic has the same syntax as $\text{MSO}(\mathcal{P}, B)$ except that the ordering relation is interpreted over the positions of a structure in Σ^* . Let $L = \bigcup \{ \text{Lin}(M) \mid M \in \mathcal{L}_\varphi \}$. We exhibit a sentence $\widehat{\varphi}$ in $\text{MSO}(\Sigma)$ such that $L = \{ \sigma \mid \sigma \models \widehat{\varphi} \}$. The main observation is that the bound B ensures that the family of channel-capacity functions $\mathcal{K}$ can be captured by a fixed number of sets, which is used both to assert channel-consistency and to express the partial order of MSCs in terms of the underlying linear order of positions. The required conclusion will then follow from Büchi's theorem [5]. $\square$

Lemma 4.2. *Let $\mathcal{L} \subseteq \mathcal{M}_{\mathcal{P},B}$ be a regular MSC language. Then $\mathcal{L}$ is $\text{MSO}(\mathcal{P}, B)$ -definable.*

Proof Sketch: Let $L = \bigcup \{ \text{Lin}(M) \mid M \in \mathcal{L} \}$. Then L is a regular (string) MSC language over Σ . Hence by Büchi's theorem [5] there exists a sentence φ in $\text{MSO}(\Sigma)$ such that $L = \{ \sigma \mid \sigma \models \varphi \}$. An important property of φ is that one linearization of an MSC satisfies φ if and only if all linearizations of the MSC satisfy φ . We then define the sentence $\widehat{\varphi} = \|\varphi\|$ in $\text{MSO}(\mathcal{P}, B)$ inductively such that the language of MSCs defined by $\widehat{\varphi}$ is precisely $\mathcal{L}$. The key idea here is to define a canonical linearization of MSCs along the lines of [20] and show that the underlying linear order is expressible in $\text{MSO}(\mathcal{P}, B)$. We obtain a formula $\widehat{\varphi}$ that says “along the canonical linearization of an MSC, the sentence φ is satisfied”.

Since $\text{MSO}(\Sigma)$ is decidable, it follows that $\text{MSO}(\mathcal{P}, B)$ is decidable as well. To conclude, we can summarize the main results of this paper as follows.

Theorem 4.3. *Let $L \subseteq \Sigma^*$, where Σ is the communication alphabet associated with a set $\mathcal{P}$ of processes. Then, the following are equivalent.*

- (i) L is a regular MSC language.
- (ii) L is a B -bounded regular MSC language, for some $B \in \mathbb{N}$.
- (iii) There exists a bounded message-passing automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L$.
- (iv) L is $\text{MSO}(\mathcal{P}, B)$ -definable, for some $B \in \mathbb{N}$.

References

1. Alur, R., Holzmann, G. J., and Peled, D.: An analyzer for message sequence charts. *Software Concepts and Tools*, **17**(2) (1996) 70–77.

2. Alur, R., and Yannakakis, M.: Model checking of message sequence charts. *Proc. CONCUR'99*, LNCS **1664**, Springer-Verlag (1999) 114–129.
3. 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.
4. Booch, G., Jacobson, I., and Rumbaugh, J.: *Unified Modeling Language User Guide*. Addison-Wesley (1997).
5. Büchi, J. R.: On a decision method in restricted second order arithmetic. *Z. Math. Logik Grundlag. Math* **6** (1960) 66–92.
6. Damm, W., and Harel, D.: LCSs: Breathing life into message sequence charts. *Proc. FMOODS'99*, Kluwer Academic Publishers (1999) 293–312.
7. Diekert, V., and Rozenberg, G. (Eds.): *The book of traces*. World Scientific (1995).
8. Ebinger, W., and Muscholl, A.: Logical definability on infinite traces. *Theoretical Computer Science* **154**(1) (1996) 67–84.
9. Harel, D., and Gery, E.: Executable object modeling with statecharts. *IEEE Computer*, July 1997 (1997) 31–42.
10. Henriksen, J. G., Mukund, M., Narayan Kumar, K., and Thiagarajan, P. S.: Towards a theory of regular MSC languages, Report RS-99-52, BRICS, Department of Computer Science, University of Aarhus, Denmark (1999).
11. Henriksen, J. G., Mukund, M., Narayan Kumar, K., and Thiagarajan, P. S.: On message sequence graphs and finitely generated regular MSC languages, *Proc. ICALP'2000*, LNCS **1853**, Springer-Verlag (2000).
12. ITU-TS Recommendation Z.120: *Message Sequence Chart (MSC)*. ITU-TS, Geneva (1997)
13. Ladkin, P. B., and Leue, S.: Interpreting message flow graphs. *Formal Aspects of Computing* **7**(5) (1995) 473–509.
14. Levin, V., and Peled, D.: Verification of message sequence charts via template matching. *Proc. TAPSOFT'97*, LNCS **1214**, Springer-Verlag (1997) 652–666.
15. Mauw, S., and Reniers, M. A.: High-level message sequence charts, *Proc. SDL '97*, Elsevier (1997) 291–306.
16. Mukund, M., Narayan Kumar, K., and Sohoni, M.: Keeping track of the latest gossip in message-passing systems. *Proc. Structures in Concurrency Theory (STRICT)*, Workshops in Computing Series, Springer-Verlag (1995) 249–263.
17. Muscholl, A.: Matching Specifications for Message Sequence Charts. *Proc. FOSACS'99*, LNCS **1578**, Springer-Verlag (1999) 273–287.
18. Muscholl, A., Peled, D., and Su, Z.: Deciding properties for message sequence charts. *Proc. FOSSACS'98*, LNCS **1378**, Springer-Verlag (1998) 226–242.
19. Rudolph, E., Graubmann, P., and Grabowski, J.: Tutorial on message sequence charts. In *Computer Networks and ISDN Systems—SDL and MSC*, Volume **28** (1996).
20. Thiagarajan, P. S., and Walukiewicz, I.: An expressively complete linear time temporal logic for Mazurkiewicz traces. *Proc. IEEE LICS'97* (1997) 183–194.
21. Thomas, W.: Automata on infinite objects. In van Leeuwen, J. (Ed.), *Handbook of Theoretical Computer Science, Volume B*, North-Holland (1990) 133–191.
22. Vardi, M. Y., and Wolper, P.: An automata-theoretic approach to automatic program verification. In *Proc. IEEE LICS'86* (1986) 332–344.
23. Zielonka, W.: Notes on finite asynchronous automata. *R.A.I.R.O.—Inf. Théor. et Appl.*, **21** (1987) 99–135.

Alternating and Empty Alternating Auxiliary Stack Automata^{*}

Markus Holzer and Pierre McKenzie

Département d'I.R.O., Université de Montréal, C.P. 6128,
succ. Centre-Ville, Montréal (Québec), H3C 3J7 Canada
email: {holzer,mckenzie}@iro.umontreal.ca

Abstract. We consider variants of alternating auxiliary stack automata and characterize their computational power when the number of alternations is bounded by a constant or unlimited. In this way we get new characterizations of **NP**, the polynomial hierarchy, **PSpace**, and bounded query classes like $\mathbf{NL}^{\langle \mathbf{NP}^{[1]} \rangle}$ and $\Theta_2\mathbf{P} = \mathbf{P}^{\mathbf{NP}^{[O(\log n)]}}$, in a uniform framework.

1 Introduction

An auxiliary pushdown automaton is a resource bounded Turing machine with a separate resource unbounded pushdown store. Probably, such machines are best known for capturing **P** when their space is logarithmically bounded [2] and for capturing the important class $\text{LOG}(\mathbf{CFL}) \subseteq \mathbf{P}$ when additionally their time is polynomially bounded [11]. These two milestones in reality form part of an extensive list of equally striking characterizations (see [13, pages 373–379]). For example, a *stack* (**S**) is a pushdown store allowing its interior content to be *read* at any time, a *nonerasing stack* (**NES**) is a stack which cannot be popped, and a *checking stack* (**CS**) is a nonerasing stack which forbids any push operation once an interior stack symbol gets read. Cook's seminal result [2] alluded to above, that

$$\mathbf{AuxPD-DSpace}(s(n)) = \mathbf{AuxPD-NSpace}(s(n)) = \bigcup \mathbf{DTime}(2^{c \cdot s(n)})$$

when $s(n) \geq \log n$, is in sharp contrast with Ibarra's [3], who proved that

$$\mathbf{AuxS-DSpace}(s(n)) = \mathbf{AuxS-NSpace}(s(n)) = \bigcup \mathbf{DTime}(2^{2^{c \cdot s(n)}}),$$

$$\mathbf{AuxNES-DSpace}(s(n)) = \mathbf{AuxNES-NSpace}(s(n)) = \bigcup \mathbf{DSpace}(2^{c \cdot s(n)}),$$

$$\mathbf{AuxCS-NSpace}(s(n)) = \bigcup \mathbf{NSpace}(2^{c \cdot s(n)}),$$

$$\mathbf{AuxCS-DSpace}(s(n)) = \mathbf{DSpace}(s(n)),$$

where unions are over c and our class nomenclature should be clear (or else refer to Section 2).

^{*} Supported by the National Sciences and Engineering Research Council (NSERC) of Canada and by the *Fonds pour la Formation de Chercheurs et l'Aide à la Recherche* (FCAR) of Québec.

In the wake of [1], pushdown stores were also added to *alternating* Turing machines [7,8]. Most notably, **AuxPD**- alternating automata were shown strictly more powerful than their deterministic counterparts, and a single alternation level in a $s(n)$ space bounded **AuxPD**- automaton was shown as powerful as any constant number. In the spirit of Sudborough's $\text{LOG}(\text{CFL})$ characterization [11], Jenner and Kirsig [5] further used **AuxPD**- alternating automata with simultaneous resource bounds to capture **PH**, the polynomial hierarchy. Subsequently, Lange and Reinhardt [9] helped shed light on why **AuxPD**- alternating automata are so powerful. They introduced of a new concept: a machine is *empty alternating* if it only alternates when all its auxiliary memories and all its tapes except a logarithmic space bounded part are empty. They showed that time bounded **AuxPD**- empty alternating automata precisely capture the AC^k -hierarchy.

Alternating auxiliary *stack* automata were also investigated. For example, Ladner *et al.* [7] showed that alternation provably adds power to otherwise non-deterministic **AuxS**- space bounded automata. Further results in the case of unbounded numbers of alternations are that **AuxS**- and **AuxNES**- space bounded automata are then equally powerful, i.e., the ability to erase the stack is in fact inessential.

The aim of the present paper is to just about complete the picture afforded by **AuxS**-, by **AuxNES**-, and by **AuxCS**- automata, in the presence of alternation or of empty alternation. We distinguish between bounded and unbounded numbers of alternations, and we consider arbitrary space bounds. In particular, we investigate **AuxCS**- alternating automata, a model overlooked by Ladner *et al.* [7]. We also completely answer the question posed by Lange and Reinhardt [9] concerning the power of variants of *empty* alternating auxiliary stack automata. More generally, we refine previous characterizations in several directions. For example, to our knowledge nothing was known about **AuxS**-, **AuxNES**-, and **AuxCS**- automata with a constant number of alternations, with running time restrictions, and/or with the feature of empty alternation. We consider these models here. Our precise results and their relationships with former work are depicted in Tables 1 and 2.

The technical depth of our results varies from immediate to more subtle extensions to previous work. Indeed a baggage of techniques has developed in the literature, in the form of a handful of fundamental “tricks” forming the basis of all known simulations (examples: Cook’s “surface configuration” trick, Ibarra’s and former authors’ “pointer to a pointer” trick, Ladner *et al.*’s tricks to simulate alternation). The difficulty, when analyzing models which combine features for which individual tricks are known, is that some of the tricks are *a priori* incompatible. A typical example arises with **AuxCS**- empty alternating automata: the standard simulation method used to bound the number of alternations clashes with the checking stack property, because the use of the checking stack and empty alternation appears to be mutually exclusive.

The paper is organized as follows: the next section contains preliminaries and in Section 3 we investigate **AuxS**-, **AuxNES**-, and **AuxCS**- alternating automata. Then Section 4 is devoted to empty alternation and finally we summarize our results and highlight a (few) remaining open questions. Some of the proofs are deferred to the journal version.

2 Definitions

We assume the reader to be familiar with the basics of complexity theory as contained in Wagner and Wechsung [13]. In particular, $\Sigma_{a(n)}\mathbf{SpaceTime}(s(n), t(n))$ ($\Pi_{a(n)}\mathbf{SpaceTime}(s(n), t(n))$, respectively) denotes the class of all languages accepted by $O(s(n))$ space and $O(t(n))$ time bounded alternating Turing machines making no more than $a(n) - 1$ alternations starting in an existential (universal, respectively) state. Thus, $a(n) = 1$ covers nondeterminism and by convention $a(n) = 0$ denotes the deterministic case. For simplicity we write **N** instead of Σ_1 and **D** for Σ_0 . Moreover, if the number of alternations is unrestricted, we simply replace $\Sigma_{a(n)}$ by **A**. If we are interested in space and time classes only, we simply write **Space**($s(n)$) and **Time**($s(n)$), respectively, in our notations. We consider: $\mathbf{L} := \mathbf{DSpace}(\log n) \subseteq \mathbf{NL} := \mathbf{NSpace}(\log n) \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSpace}$. In particular, $\Sigma_k\mathbf{P}$ and $\Pi_k\mathbf{P}$, for $k \geq 0$, denote the classes of the polynomial hierarchy.

In the following we consider Turing machines with two-way input equipped with an auxiliary pushdown, stack, nonerasing stack, and checking stack storage. A *pushdown* (**PD**) is a last-in first-out (LIFO) storage structure, which is manipulated by pushing and popping. By *stack storage*, we mean a *stack* (**S**), *nonerasing stack* (**NES**), or *checking stack* (**CS**) as defined earlier. The class of languages accepted by $O(s(n))$ space bounded alternating Turing machines with auxiliary stack is denoted by **AuxS-ASpace**($s(n)$). The infix **S** is changed to **NES** (**CS**, respectively) if we consider Turing machines with auxiliary nonerasing stack (checking stack, respectively) storage. Deterministic, nondeterministic, bounded alternating classes, and simultaneously space and time bounded classes are appropriately defined and denoted.

For Turing machines augmented with an auxiliary type X storage, the concept of empty alternation was introduced in the context of logspace by Lange and Reinhardt [9]. More precisely, we define an auxiliary storage automaton to be empty alternating if in moments of alternation, i.e., during transitions between existential and universal configurations and *vice versa*, the auxiliary storage is empty and all transferred information is contained in the state and on the $s(n)$ space bounded Turing tape. We indicate that a class is defined by empty alternation by inserting a letter **E** in front of $\Sigma_{a(n)}$, $\Pi_{a(n)}$, or **A**. Thus, e.g., the class of all languages accepted by empty $s(n)$ space bounded alternating Turing machines with an auxiliary storage of type X is denoted by **AuxX-EASpace**($s(n)$).

Finally, we need some more notations on relativized complexity classes, especially on adaptive, non-adaptive, and bounded query classes. For a class **C** of languages let $\mathbf{DTime}(t(n))^{\mathbf{C}}$ be the class of all languages accepted by de-

terministic $O(t(n))$ time bounded Turing machines using an oracle $B \in \mathbf{C}$. If the underlying oracle Turing machine is nondeterministic, we distinguish between Ladner-Lynch (LL) [6] and Ruzzo-Simon-Tompa (RST) [10] relativization. In the latter, the oracle tape is written deterministically and is denoted by $\mathbf{NTime}(t(n))^{(\mathbf{C})}$. It is well known that the polynomial hierarchy may be characterized in terms of oracle access, i.e., $\Sigma_0 \mathbf{P} = \mathbf{P}$ and $\Sigma_{k+1} \mathbf{P} = \mathbf{NP}^{\Sigma_k \mathbf{P}}$ for $k \geq 0$. By definition, the Turing machine may compute the next query depending on the answers to the previous queries, i.e., it makes adaptive queries. We define the “non-adaptive query class” $\mathbf{DTime}(t(n))_{||}^{\mathbf{C}}$ to be the class of languages accepted by some deterministic $O(t(n))$ time bounded Turing machine M using an oracle $B \in \mathbf{C}$ such that for all inputs x and all computations on x , a list of all queries is formed before any of them is made. We say that M makes parallel queries. Moreover, for a function $r : \mathbb{N} \rightarrow \mathbb{N}$ and a class $\mathbf{C}$ let $\mathbf{DTime}(t(n))_{||}^{\mathbf{C}[r]}$ denote the class of all languages accepted by some deterministic $O(t(n))$ time bounded Turing machine M using an oracle $B \in \mathbf{C}$ such that, for all inputs x and all computations of M on x , the number of queries to the oracle B is less than or equal $r(|x|)$. For further explanation and results on bounded query classes we refer to [14].

Obviously, all these definitions concerning oracle Turing machines and classes can be adapted in a straight-forward manner to space bounded Turing machines—there the oracle tape is not subject to the space bound. The class $\Theta_2 \mathbf{P}$ is defined as $\mathbf{L}^{\mathbf{NP}}$. It equals $\mathbf{L}^{\mathbf{NP}[O(\log n)]}$, $\mathbf{P}^{\mathbf{NP}[O(\log n)]}$, $\mathbf{L}_{||}^{\mathbf{NP}}$, and $\mathbf{P}_{||}^{\mathbf{NP}}$ [14, page 844, Theorem 8.1], contains $\Sigma_1 \mathbf{P} \cup \Pi_1 \mathbf{P} = \mathbf{NP} \cup \mathbf{co-NP}$ and is contained in $\Sigma_2 \mathbf{P} \cap \Pi_2 \mathbf{P}$.

3 Alternating Auxiliary Stack Automata

In this section we consider alternating auxiliary stack automata with and without runtime restrictions.

3.1 Automata without Runtime Restrictions

Ladner *et al.* [7] showed $\mathbf{AuxS-ASpace}(s(n)) = \mathbf{AuxNES-ASpace}(s(n)) = \bigcup_c \mathbf{DTime}(2^{2^{c \cdot s(n)}})$ if $s(n) \geq \log n$. Surprisingly, auxiliary checking stacks were not considered—not even mentioned—in this paper. We complete the picture by showing that when the number of alternations is unbounded, $s(n)$ space bounded alternating checking stacks are as powerful as stacks and nonerasing stacks. Due to the lack of space we omit the proof of the following theorem, which follows the lines of Ladner *et al.* [7, page 152, Theorem 5.1].

Theorem 1. $\mathbf{AuxX-ASpace}(s(n)) = \bigcup_c \mathbf{DTime}(2^{2^{c \cdot s(n)}})$ if X is a stack storage and $s(n) \geq \log n$. $\square$

Now, let us take a look at variants of alternating auxiliary stack automata with a bounded number of alternations.

Theorem 2. $\text{AuxX-}\Sigma_k\text{Space}(s(n)) \subseteq \bigcup_c \text{NSpace}(2^{c \cdot s(n)})$ if X is a stack storage, $k \geq 1$, and $s(n) \geq \log n$.

Sketch of Proof. Recall that the alternation bounded PDA Theorem of Ladner *et al.* [8, page 100–104, Section 5] shows that for constant k and $s(n) \geq \log n$, $\text{AuxPD-}\Sigma_k\text{Space}(s(n)) \subseteq \bigcup_c \text{NSpace}(2^{c \cdot s(n)})$. The main idea in the lengthy argument used by Ladner *et al.* to prove their theorem is to generalize the notion of realizable pairs of surface configurations. A pair (ID, U) , where ID is a surface configuration ID and U a set of (popping) surface configurations, is realizable if there is a computation tree (including all children at a universal node and precisely one child at an existential node) whose root is labeled ID , whose leaves have labels which are members of U , ID and all surface configurations in U have the same pushdown height, and the height of the pushdown does not go below this level during the computation. The definition of a realizable pair (ID, U) gives rise to a recursive nondeterministic algorithm, which is roughly bounded by the space to store a set of surface configurations, that is $2^{c \cdot s(n)}$ for some constant c .

In case of an auxiliary stack automaton we can not use this algorithm directly, but we may adapt it for our needs. Since the major difference between a pushdown and a stack is that the latter is also allowed to read but not to change the interior content, we have to face and overcome this problem. When in read mode, an alternating auxiliary stack automaton M can be viewed as an alternating *finite* automaton with $2^{c \cdot s(n)}$ states for some constant c (reading the stack content). Thus, the behaviour of M in this mode can be described by a table whose entries are sets of surface configurations. This idea was also used by several authors, see, e.g., Ladner *et al.* [7, page 148], where the tables are called *response sets*. An entry (ID, U) in the table T means that there is a computation tree whose root is labeled ID , all leaves have labels in U and entering push or pop mode, and the whole computation is done in read mode. Obviously, there are at most $2^{c' \cdot s(n)}$ tables for some constant c' . We call a pair (ID, T) an extended surface configuration if ID is an ordinary surface configuration and T is a table as described above.

Now we alter the algorithm of Ladner *et al.* for auxiliary pushdown automata, so that it works on extended surface configurations and sets, by adapting the derivation relations accordingly. A careful analysis shows that it may be implemented on a Turing machine with a constant number of alternations. The space bound is bounded by the number of extended surface configurations which is roughly $2^{c'' \cdot s(n)}$ for some constant c'' . Since the constant bounded alternation hierarchy for space bounded Turing machines collapses by Immerman [4] and Szelepcsényi [12] we obtain our result. $\square$

3.2 Automata with Restricted Running Time

Now let's turn our attention to simultaneous space and time bounds. There we find the following situation, which can be shown by a phase-by-phase simulation,

preserving the number of alternations on both sides. Due to the lack of space we omit the details.

Theorem 3. $\bigcup_c \mathbf{AuxX-ASpaceTime}(s(n), 2^{c \cdot s(n)}) = \bigcup_c \mathbf{ATime}(2^{c \cdot s(n)})$ if X is a stack storage and $s(n) \geq \log n$. $\square$

Since $\mathbf{PSpace} = \mathbf{ATime}(\text{pol } n)$ we conclude:

Corollary 4. $\bigcup_c \mathbf{AuxX-ASpaceTime}(\log n, \text{pol } n) = \mathbf{PSpace}$ if X is a stack storage. $\square$

Since the simulation for the inclusion from right to left in the previous proof uses at least a nondeterministic auxiliary stack automaton we can state:

Corollary 5. $\bigcup_c \mathbf{AuxX-}\Sigma_k\mathbf{SpaceTime}(s(n), 2^{c \cdot s(n)}) = \bigcup_c \Sigma_k\mathbf{Time}(2^{c \cdot s(n)})$ if X is a stack storage, $k \geq 1$, and $s(n) \geq \log n$. $\square$

But what about the deterministic case? The following theorem answers this question for auxiliary stack and nonerasing stack automata and can be shown by step-by-step simulations.

Theorem 6. Let $s(n) \geq \log n$. We have:

1. $\bigcup_c \mathbf{AuxX-DSpaceTime}(s(n), 2^{c \cdot s(n)}) = \bigcup_c \mathbf{DTime}(2^{c \cdot s(n)})$ if X is a stack or nonerasing stack.
2. $\bigcup_c \mathbf{AuxCS-DSpaceTime}(s(n), 2^{c \cdot s(n)}) = \mathbf{DSpace}(s(n))$. $\square$

The results in this subsection in the case $s(n) = \log n$ yield an alternate characterization of the polynomial hierarchy, to be contrasted with that given by Jenner and Kirsig [5, page 123, Theorem 3.4] in terms of auxiliary pushdown automata, where $\mathbf{AuxPD-}\Sigma_{k+1}\mathbf{SpaceTime}(\log n, \text{pol } n) = \Sigma_k\mathbf{P}$ for $k \geq 1$ was shown. Note that [5] claims the case $k \geq 1$ of part 2 of the following corollary.

Corollary 7. 1. $\mathbf{AuxX-}\Sigma_k\mathbf{SpaceTime}(\log n, \text{pol } n) = \Sigma_k\mathbf{P}$ if X is a stack or nonerasing stack and $k \geq 0$.
 2. $\mathbf{AuxCS-}\Sigma_0\mathbf{SpaceTime}(\log n, \text{pol } n) = \mathbf{DSpace}(\log n)$ and the checking stack class $\mathbf{AuxCS-}\Sigma_k\mathbf{SpaceTime}(\log n, \text{pol } n) = \Sigma_k\mathbf{P}$ if $k \geq 1$. $\square$

4 Empty Alternating Auxiliary Stack Automata

Lange and Reinhardt [9] exhibited a close connection between (RST) relativized complexity classes and empty alternation. Here we obtain similar results for empty alternating auxiliary stack automata. We start with a lemma generalizing a result of Lange and Reinhardt [9, page 501, Theorem 10].

Lemma 8. Let X be a stack storage and $s(n) \geq \log n$. If $\mathbf{AuxX-NSpace}(s(n)) \subseteq \mathbf{C}_1$ and $\bigcup_c \mathbf{AuxX-NSpaceTime}(s(n), 2^{c \cdot s(n)}) \subseteq \mathbf{C}_2$, for some $\mathbf{C}_1$ and $\mathbf{C}_2$, then

1. $\mathbf{AuxX-EASpace}(s(n)) \subseteq \bigcup_c \mathbf{DTime}(2^{c \cdot s(n)})_{||}^{\mathbf{C}_1}$ and
2. $\bigcup_c \mathbf{AuxX-EASpaceTime}(s(n), 2^{c \cdot s(n)}) \subseteq \bigcup_c \mathbf{DTime}(2^{c \cdot s(n)})_{||}^{\mathbf{C}_2}$. $\square$

4.1 Automata without Runtime Restrictions

With the help of the above lemma we can now extend the argument in [9, Page 498, Theorem 5] and show that the empty alternation hierarchy for auxiliary stack automata collapses to its nondeterministic level—we omit the proof.

Theorem 9. $\mathbf{AuxX-EASpace}(s(n)) = \mathbf{AuxX-NSpace}(s(n))$ if X is a stack storage and $s(n) \geq \log n$. $\square$

For $s(n) = \log n$ we obtain from Theorem 9 the following corollary.

Corollary 10. 1. $\mathbf{AuxS-EASpace}(\log n) = \bigcup_k \mathbf{DTime}(2^{n^k})$ and
 2. $\mathbf{AuxX-EASpace}(\log n) = \mathbf{PSpace}$ if X is a nonerasing or checking stack. $\square$

4.2 Automata with Restricted Running Time

Obviously, for empty alternating Σ_0 and Σ_1 machines we profit from the results in Subsection 3.2 and refer in these cases to Corollary 5 and Theorem 6. Moreover, for an unbounded number of alternations the upper bound was already settled in Lemma 8. We start with auxiliary stack automata and adapt an argument from [9, page 501, Theorem 10] proving a lower bound.

Lemma 11. Let X be a stack storage. If $s(n) \geq \log n$ and $\mathbf{C}$ is contained in $\bigcup_c \mathbf{AuxX-NSpaceTime}(s(n), 2^{c \cdot s(n)})$, then $\mathbf{DSpace}(s(n))^{\mathbf{C}[O(s(n))]}$ is a subset of $\bigcup_c \mathbf{AuxX-E}\Sigma_k \mathbf{SpaceTime}(s(n), 2^{c \cdot s(n)})$, where $k = 2$ if X is a stack and $k = 3$ if X is a nonerasing or checking stack storage. $\square$

The above lemma together with Lemma 8 show that for $s(n) \geq \log n$ we have sandwiched $\mathbf{AuxX-E}\Sigma_k \mathbf{SpaceTime}(s(n), 2^{c \cdot s(n)})$ in between the classes $\mathbf{DSpace}(s(n))^{\mathbf{C}[O(s(n))]}$ and $\bigcup_c \mathbf{DTime}(2^{c \cdot s(n)})_{||}^{\mathbf{C}}$, with $\mathbf{C} = \bigcup_c \mathbf{NTime}(2^{c \cdot s(n)})$ because of Corollary 5. In case $s(n) = \log n$, these bounded query classes are known to be equal due to Wagner [14, page 838, Corollary 3.7.1]. Fortunately, this generalizes to arbitrary space bounds greater or equal than $\log n$.

Theorem 12. $\mathbf{DSpace}(s(n))^{\mathbf{C}[O(s(n))]} = \bigcup_c \mathbf{DTime}(2^{c \cdot s(n)})_{||}^{\mathbf{C}}$ if $s(n) \geq \log n$ and $\mathbf{C} = \bigcup_c \mathbf{NTime}(2^{c \cdot s(n)})$. $\square$

As an immediate corollary we obtain:

Corollary 13. Let X be a stack storage. If $s(n) \geq \log n$ and $\mathbf{C}$ is equal to $\bigcup_c \mathbf{NTime}(2^{c \cdot s(n)})$, then $\bigcup_c \mathbf{AuxX-E}\Sigma_2 \mathbf{SpaceTime}(s(n), 2^{c \cdot s(n)})$ and the class $\bigcup_c \mathbf{AuxX-EASpaceTime}(s(n), 2^{c \cdot s(n)})$, where $k = 2$ if X is a stack and $k = 3$ if X is a nonerasing or checking stack, coincide with $\bigcup_c \mathbf{DTime}(2^{c \cdot s(n)})_{||}^{\mathbf{C}}$. $\square$

For the special case $s(n) = \log n$ the above corollary results in a characterization of $\Theta_2 \mathbf{P}$, using $\Theta_2 \mathbf{P} = \mathbf{P}_{||}^{\mathbf{NP}} = \mathbf{LNP}^{[O(\log n)]} = \mathbf{PNP}^{[O(\log n)]}$ proven by Wagner [14, page 844, Theorem 8.1]. A similar result was obtained by Lange and Reinhardt [9, page 501, Theorem 10] for empty alternating polytime machines.

Corollary 14. $\Theta_2\mathbf{P}$ is equal to $\mathbf{AuxX-E}\Sigma_k\mathbf{SpaceTime}(\log n, \text{pol } n)$ and the class $\mathbf{AuxX-EASpaceTime}(\log n, \text{pol } n)$, where X is a stack storage and $k = 2$ if X is a stack and $k = 3$ if X is a nonerasing or checking stack. $\square$

Thus, it remains to classify one empty alternation on auxiliary nonerasing and checking stack automata.

Theorem 15. $\bigcup_c \mathbf{AuxX-E}\Sigma_2\mathbf{SpaceTime}(s(n), 2^{c \cdot s(n)}) = \mathbf{NSpace}(s(n))^{(\mathbf{C}[1])}$ if X is a nonerasing or checking stack storage, $s(n) \geq \log n$, and $\mathbf{C}$ is equal to $\mathbf{AuxX-NSpaceTime}(s(n), 2^{c \cdot s(n)})$.

Proof. First we prove the inclusion from right to left. Let M be a nondeterministic $s(n)$ space bounded Turing machine with an oracle $A \in \mathbf{C}$ making exactly one oracle question. Further, let M_A and $M_{\bar{A}}$ be the auxiliary automaton accepting A and its complement, respectively. Observe that $M_{\bar{A}}$ is a machine with universal states only.

We simply simulate M 's nondeterministic moves by an auxiliary nondeterministic automaton M' with storage X . If M starts deterministically writing on the oracle tape, the ID of M is stored on the space bounded worktape, M' guesses the answer to that oracle question, stores it in the finite control, and proceeds with the simulation of M . If M accepts, the verification of the oracle answers has to be done. In case the answer was “yes,” the machine M starts the simulation of M_A using ID to deterministically reconstruct the oracle question. If M_A accepts, then M' accepts. The “no” answer is checked by M by universally alternating—the stack storage is empty, because it was not used up to now—and simulating $M_{\bar{A}}$. Here M' accepts, if $M_{\bar{A}}$ does. Thus, two empty alternations suffice.

For the converse inclusion let M be a $s(n)$ space and $2^{c \cdot s(n)}$ time bounded empty alternating Σ_2 auxiliary nonerasing (checking) stack automaton. Let $C(x)$ be the set of all configurations of M on input x with empty auxiliary nonerasing (checking) stack storage. Define the oracles

$$A_{\exists, M} = \{ ID \in C(x) \mid M \text{ starting in } ID \text{ accepts with existential moves only} \}$$

and

$$A_{\forall, M} = \{ ID \in C(x) \mid M \text{ starting in } ID \text{ accepts with universal moves only} \}.$$

For the simulation we consider two cases. Either M writes the first time to the stack in the existential or universal phase. Note that in the former case M can not alternate anymore. Thus, the simulation by a $s(n)$ space bounded nondeterministic Turing machine M' runs as follows: Machine M' nondeterministically guesses whether the first write instruction is performed during the existential or universal phase of M . In the former case, M' asks the question “ $ID_0 \in A_{\exists, M}$,” where ID_0 is the initial configuration of M , and accepts if the answer is “yes.” In the latter case, where the stack is not used in existential moves, M' simulates M step-by-step until it reaches the ID where the alternation appears, and asks the complement of $A_{\forall, M}$. If on question “ $ID \in \text{co-}A_{\forall, M}$ ” the answer is “yes,” then M' rejects. Otherwise, it accepts. Since $A_{\exists, M}$ and complement of $A_{\forall, M}$ both belong to $\mathbf{C}$ our claim follows. $\square$

For our favourite space bound $s(n) = \log n$ this results in:

Corollary 16. $\mathbf{NL}^{(\mathbf{NP}[1])} = \mathbf{AuxX-E}\Sigma_2\mathbf{SpaceTime}(\log n, \text{pol } n)$ if X is a non-erasing or checking stack. $\square$

5 Conclusions

Tables 1 and 2 summarize the known complexity class characterizations based on **AuxS**-, **AuxNES**-, and **AuxCS**- alternating and empty-alternating automata. In addition to the short-hand notations introduced in Section 2 we use **EDTime** (**EEDTime**, respectively) to denote the class $\bigcup_k \mathbf{DTime}(2^{n^k})$ ($\bigcup_k \mathbf{DTime}(2^{2^{n^k}})$, respectively) and also **ENSpace** as an abbreviation for the class $\bigcup_k \mathbf{NSpace}(2^{n^k})$.

Table 1. Complexity classes of auxiliary $\log n$ space bounded alternating automata (shading represents results obtained in this paper)

Alternations	Auxiliary alternating Turing machine with storage X								
	none	PD-		S-		NES-		CS-	
		time bound		time bound		time bound		time bound	
		poly	without	poly	without	poly	without	poly	without
determ.	L	$\text{LOG}(\mathbf{DCFL})$ [11]	P [2]	P	EDTime [3]	P	PSpace [3]	L	L [3]
Σ_1	NL	$\text{LOG}(\mathbf{CFL})$ [11]	PSpace [8]	NP	EDTime $\subseteq$ $\cdot$ $\subseteq$ ENSpace	NP	PSpace $\subseteq$ $\cdot$ $\subseteq$ ENSpace	NP	PSpace [3]
Σ_2		NP [5]		$\Sigma_2 \mathbf{P}$		$\Sigma_2 \mathbf{P}$		$\Sigma_2 \mathbf{P}$	PSpace
Σ_3		$\Sigma_2 \mathbf{P}$ [5]		$\Sigma_3 \mathbf{P}$		$\Sigma_3 \mathbf{P}$		$\Sigma_3 \mathbf{P}$	$\cdot$
$\cdot$		$\cdot$		$\cdot$		$\cdot$		$\cdot$	$\subseteq$
$\cdot$		$\cdot$		$\cdot$		$\cdot$		$\cdot$	$\subseteq$
Σ_k	[4, 12]	$\Sigma_{k-1} \mathbf{P}$ [5]		$\Sigma_k \mathbf{P}$		$\Sigma_k \mathbf{P}$		$\Sigma_k \mathbf{P}$	ENSpace
$\cdot$		$\cdot$		$\cdot$		$\cdot$		$\cdot$	$\cdot$
$\cdot$		$\cdot$		$\cdot$		$\cdot$		$\cdot$	$\cdot$
A	P [1]	PSpace [5]	EDTime [7]	PSpace	EEDTime [7]	PSpace	EEDTime [7]	PSpace	EEDTime

The picture is complete in the case of empty alternating automata. In the case of alternating automata, the picture is complete, with only three exceptions involving bounded numbers of alternation greater than one (table entry $\mathbf{A} \subseteq \cdot \subseteq \mathbf{B}$). These entries indicate that the corresponding space-bounded **AuxS**-, **AuxNES**-, and **AuxCS**- alternating automata classes are somewhere between **A** and **B**, but what is their precise status?

Table 2. Complexity classes of auxiliary log n space bounded *empty* alternating automata (shading represents results obtained in this paper)

Alternations	Auxiliary <i>empty</i> alternating Turing machine with storage X								
	none	PD-		S-		NES-		CS-	
		time bound		time bound		time bound		time bound	
		poly	without	poly	without	poly	without	poly	without
determ.	L	LOG(DCFL) [11]	P [2]	P	EDTime [3]	P	PSpace [3]	L	L [3]
$E\Sigma_1$	NL	LOG(CFL) [11]		NP		NP		NP	PSpace [3]
$E\Sigma_2$		LOG(CFL) [9]	P [9]	$\Theta_2 P$	EDTime	NL(NP ^[1])	PSpace	NL(NP ^[1])	PSpace
$E\Sigma_3$						$\Theta_2 P$		$\Theta_2 P$	
⋮									
$E\Sigma_k$	[4, 12]								
⋮									
EA	P [1]	P [9]							

References

1. A. K. Chandra, D. C. Kozen, and L. J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981.

2. S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, 1971.

3. O. H. Ibarra. Characterizations of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata. *Journal of Computer and System Sciences*, 5(2):88–117, 1971.

4. N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.

5. B. Jenner and B. Kirsig. Characterizing the polynomial hierarchy by alternating auxiliary pushdown automata. In *Proceedings of the 5th Annual Symposium on Theoretical Computer Science*, number 294 in LNCS, pages 118–125, Bordeaux, France, 1988. Springer.

6. R. Ladner and N. Lynch. Relativization of questions about log space computability. *Mathematical Systems Theory*, 10:19–32, 1976.

7. R. E. Ladner, R. J. Lipton, and L. J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13(1):135–155, 1984.

8. R. E. Ladner, L. J. Stockmeyer, and R. J. Lipton. Alternation bounded auxiliary pushdown automata. *Information and Control*, 62:93–108, 1984.

9. K.-J. Lange and K. Reinhardt. Empty alternation. In *Proceedings of the 9th Conference on Mathematical Foundations of Computer Science*, number 841 in LNCS, pages 494–503, Kosice, Slovakia, 1994. Springer.

10. W. L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computations. *Journal of Computer and System Sciences*, 28(2):216–230, 1984.

11. I. H. Sudborough. On the tape complexity of deterministic context-free languages. *Journal of the ACM*, 25(3):405–414, 1978.

12. R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26(3):279–284, 1988.

13. K. Wagner and G. Wechsung. *Computational Complexity*. Mathematics and its applications (East Europeans series). VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.
14. K. W. Wagner. Bounded query classes. *SIAM Journal on Computing*, 19(5):833–846, 1990.

Counter Machines: Decidable Properties and Applications to Verification Problems

Oscar H. Ibarra, Jianwen Su, Zhe Dang, Tevfik Bultan, and Richard Kemmerer

Department of Computer Science, University of California, Santa Barbara, CA 93106, USA.
{ibarra, su, dang, bultan, kemm}@cs.ucsb.edu

Abstract. We study various generalizations of reversal-bounded multicounter machines and show that they have decidable emptiness, infiniteness, disjointness, containment, and equivalence problems. The extensions include allowing the machines to perform linear-relation tests among the counters and parameterized constants (e.g., “Is $3x - 5y - 2D_1 + 9D_2 < 12$?”, where x, y are counters, and D_1, D_2 are parameterized constants). We believe that these machines are the most powerful machines known to date for which these decision problems are decidable. Decidability results for such machines are useful in the analysis of reachability problems and the verification/debugging of safety properties in infinite-state transition systems. For example, we show that (binary, forward, and backward) reachability, safety, and invariance are solvable for these machines.

1 Introduction

The simplest language recognizers are the finite automata. It is well known that all varieties of finite automata (one-way, two-way, nondeterministic, etc.) are effectively equivalent, and the class has decidable emptiness, infiniteness, disjointness, containment, and equivalence problems. These problems, referred to as *F-problems*, are defined as follows, for arbitrary finite automata M_1, M_2 :

- *Emptiness*: Is $L(M_1)$ (the language accepted by M_1) empty?
- *Infiniteness*: Is $L(M_1)$ infinite?
- *Disjointness*: Is $L(M_1) \cap L(M_2)$ empty?
- *Containment*: Is $L(M_1) \subseteq L(M_2)$?
- *Equivalence*: Is $L(M_1) = L(M_2)$?

When a two-way finite automaton is augmented with a storage device, such as a counter, a pushdown stack or a Turing machine tape, the F-problems become undecidable (no algorithms exist). In fact, it follows from a result in [12] that the emptiness problem is undecidable for two-way counter machines even over a unary input alphabet. On binary inputs, if one restricts the counter machines to make only a finite number of turns on the input tape, the emptiness problem is also undecidable, even for the case when the input head makes only one turn (i.e., change in direction) [9]. However, for one-way counter machines, it is known that the equivalence (hence also the emptiness) problem is decidable, but the containment and disjointness problems are undecidable [14].

In this paper, we study two-way finite automata augmented with finitely many counters. A restricted version of these machines was studied in [9], where: i) each counter is

reversal-bounded in that it can be incremented or decremented by 1 and tested for zero, but the number of times it can change mode from nondecreasing to nonincreasing and vice-versa is bounded by a constant, and ii) the two-way input is finite-crossing in that the number of times the input head crosses the boundary between any two adjacent cells of the input tape is bounded by a constant (there is no bound on how long the head can remain on a cell).

We consider various generalizations of finite-crossing reversal-bounded multicounter machines and investigate their decision problems. The extensions include allowing the machines to perform linear-relation tests among the counters and parameterized constants (e.g., “Is $3x - 5y - 2D_1 + 9D_2 < 12$?”, where x, y are counters and D_1, D_2 are parameterized constants). We show that many classes have decidable F-problems. We believe that these machines are the most powerful machines known to date for which the decision problems are decidable. Decidability results for such machines are useful in the analysis of reachability problems and the verification/debugging of safety properties in infinite-state transition systems. For example, we show that (binary, forward, and backward) reachability, safety, and invariance are solvable for these machines.

2 Reversal-Bounded Multicounter Machines

It is convenient to represent a counter machine as a program. The standard model of a deterministic two-way multicounter machine can be specified by a program M of the form

begin input ($\#w\#$); P **end**.

Here w is the input with delimiters $\#$, and P is a sequence of labeled instructions, where each instruction is of the form shown in Fig. 1, where (1) s, p, q denote labels or states (we will use the latter terminology in the paper); (2) **read** ($INPUT$) means read the symbol currently under the input head and store it in $INPUT$; (3) a is $\#$ or a symbol in the input alphabet of the machine; (4) The instruction **left** means move the input head one cell to the left, and **right** means move the input head one cell to the right; (5) x represents a counter. Thus a machine with k counters will have k such x 's.

$s : \text{read } (INPUT)$	$s : x := x + 1$	$s : \text{left}$	$s : \text{accept}$
$s : \text{goto } p$	$s : x := x - 1$	$s : \text{right}$	$s : \text{reject}$
$s : \text{if } INPUT = a \text{ then goto } p \text{ else goto } q$		$s : \text{if } x = 0 \text{ then goto } p \text{ else goto } q$	

Fig. 1. Instructions

The machine starts its computation with the first instruction in P with the input head on the left delimiter and all the counters set to zero. An input $\#w\#$ is accepted (rejected) if M on this input halts in **accept** (**reject**). Note that the machine may not always halt. The set of all inputs accepted by M is denoted by $L(M)$.

We can make the machine nondeterministic by allowing a nondeterministic instruction of the form “ $s : \text{goto } p \text{ or goto } q$.” Clearly this is the only nondeterministic instruction we need. Other forms of nondeterminism (e.g., allowing nondeterministic assignments like “ $x := x + 1 \text{ or } y := y - 1$ ” or allowing instructions like “**left or right**” do not add any more power to the machine.

M is *reversal-bounded* if there is a nonnegative integer r such that for any computation on any input, every counter of M makes no more than r reversals (alternations between nondecreasing and nonincreasing modes). So, for example, a counter with the computation pattern “00000111111222222344444” has 0 reversals. On the other hand, “00000111111222222344444333222123344” has 2 reversals.

M is *finite-crossing* if there is a positive integer m such that on every computation on any input, M 's input head crosses the boundary between any two adjacent tape cells at most m times. Note that there is no bound on the number of turns the input head makes on the tape. There is also no bound on how long the head can remain (*sit*) on a symbol. M is *one-way* if it is 1-crossing.

3 Fundamental Decidable Problems

We begin with the following theorem in [9].

Theorem 1. The emptiness problem is decidable for nondeterministic one-way reversal-bounded multicounter machines.

Nondeterministic finite-crossing machines can be converted to one-way (this is *not* true for the deterministic case). Hence:

Theorem 2. The emptiness problem is decidable for nondeterministic finite-crossing reversal-bounded multicounter machines.

We can generalize Theorem 1 to allow one of the counters to be unrestricted:

Theorem 3. The emptiness problem is decidable for nondeterministic one-way machines with one unrestricted counter and several reversal-bounded counters.

Theorem 4. The infiniteness and disjointness problems are decidable for nondeterministic finite-crossing reversal-bounded multicounter machines.

Containment and equivalence are undecidable for nondeterministic machines. In fact, it is undecidable to determine, given a nondeterministic one-way machine with one 1-reversal counter, whether it accepts all strings [2]. However for deterministic machines, we can prove:

Theorem 5. The containment and equivalence problems are decidable for deterministic finite-crossing reversal-bounded multicounter machines.

4 Generalizations

4.1 Constant Increments and Comparisons

The first generalization of a multicounter machine is to allow the counters to store negative numbers, and allow the program to use assignments of the form $s : x := x + c$, and conditionals (if statements) of the form “ $s : \text{if } x\theta c \text{ then goto } p \text{ else goto } q$,” where c is an integer constant (there are a finite number of such constants in the program), and θ is one of $<$, $>$, $=$. One can easily show that any (reversal-bounded) multicounter machine M that uses these generalized instructions can be converted to an equivalent (reversal-bounded) standard model M' such that $L(M) = L(M')$.

4.2 Linear Conditions

We can further allow tests like “ $s : \text{if } 5x - 3y + 2z < 7 \text{ then goto } p \text{ else goto } q.$ ” To be precise, let V be a finite set of variables over integers. An *atomic linear relation* on V is defined as $\sum_{v \in V} a_v v < b$, where a_v and b are integers. A *linear relation* on V is constructed from a finite number of atomic linear relations using $\neg$ and $\wedge$. Note that standard symbols like $>, =, \rightarrow$ (implication), $\vee$ can also be expressed using the above constructions.

We can allow a multicounter machine M to use conditionals (tests) of the form “ $s : \text{if } L \text{ then goto } p \text{ else goto } q.$ ” where L is a linear relation on the counters. Unfortunately, the halting (and, hence, the emptiness) problem is undecidable for reversal-bounded multicounter machines that allow linear-relation conditionals. In fact, the undecidability holds even in the case of only 3 counters:

Theorem 6. Consider only deterministic machines with 3 counters, C_1, C_2 , and T , with no input tape. The counters which are initially 0 can only use instructions of the form $x := x + 1$ (where x is a counter), and linear test $T = C_1?$ or $T = C_2?$ (Note that $C_1 = C_2?$ is not allowed). The halting problem for such machines is undecidable.

Proof. A close look at the proof of the undecidability of the halting problem for two-counter machines (with no input tape) in [12] reveals that the counters behave in a regular pattern. The two counter machine operates in phases in the following way. Let C_1 and C_2 be its counters. Then M 's operation can be divided into phases $P_1, P_2, P_3, \dots$, where each P_i starts with one of the counters equal to zero and the other counter equal to some positive integer d_i . During the phase, the first counter is increasing, while the second counter is decreasing. The phase ends with the first counter having value d_{i+1} and the second counter having value 0. Then in the next phase the modes of the counters are interchanged. Thus, a sequence of configurations corresponding to the phases above will be of the form

$$(q_1, 0, d_1), (q_2, d_2, 0), (q_3, 0, d_3), (q_4, d_4, 0), \dots$$

where the q_i are states and $d_1 = 1, d_2, d_3, \dots$ are positive integers. Note that the second component of the configuration refers to the value of C_1 , while the third component refers to the value of C_2 .

We construct a 3-counter machine M' with counters C'_1, C'_2 and T which simulates M . The sequence of configurations of M' corresponding to the above phases would have the form (the second, third, and fourth components correspond to the values of C'_1, C'_2 , and T , respectively):

$$\begin{aligned} &(q_1, 0, d_1, 0), \\ &(q_2, d_1 + d_2, d_1, d_1), \\ &(q_3, d_1 + d_2, d_1 + d_2 + d_3, d_1 + d_2), \\ &(q_4, d_1 + d_2 + d_3 + d_4, d_1 + d_2 + d_3, d_1 + d_2 + d_3), \\ &(q_5, d_1 + d_2 + d_3 + d_4, d_1 + d_2 + d_3 + d_4 + d_5, d_1 + d_2 + d_3 + d_4), \dots \end{aligned}$$

To go from, for example, $(q_1, 0, d_1, 0)$ to $(q_2, d_1 + d_2, d_1, d_1)$, C'_1 and T are incremented until $T = C'_2$. During the phase, C'_1 also simulates C_1 , adding d_2 to the counter. Thus C'_1 will have value $d_1 + d_2$ at the end of the phase. ■

The 3 counters in the result above are necessary since we can show:

Theorem 7. The emptiness problem is decidable for one-way nondeterministic machines with two reversal-bounded counters, where in addition to standard instructions, the machines can use tests of the form $x\theta c$ and $x - y\theta c$ where x, y represent the two counters, c represents a constant, and θ is $>$, $<$, or $=$.

There is a stronger notion of reversal-boundedness. A counter with the computation pattern “000001111122222344444” corresponds to 0-reversal.

In this example there are segments of the computation when the counter value does not change. We define a stronger notion of reversal-boundedness. We say that M is *strongly reversal-bounded* if there is a nonnegative integer r such that for any computation on any input, every counter of M makes no more than r alternations between increasing, no-change, and decreasing modes. In the above example, the pattern corresponds to 6 strong reversals.

Obviously a strongly reversal-bounded multicounter machine is reversal-bounded. However, a reversal-bounded machine need not be strongly reversal-bounded. For example, the patterns of the form “122334455 . . .” correspond to 0-reversal, but are not strongly reversal-bounded.

Note that while the machine M' in the construction in Theorem 6 is reversal-bounded, it is not strongly reversal-bounded. However, we can prove the following:

Theorem 8. The emptiness problem is decidable for nondeterministic finite-crossing strongly reversal-bounded multicounter machines using linear-relation conditionals on the counters.

Before we give the proof we need some definitions and notations. Suppose M is a nondeterministic finite-crossing strongly reversal-bounded multicounter machine. During a computation, each counter of M can be in any of the following three modes: *increasing*, *no-change*, *decreasing*. A counter makes a mode-change if it goes from mode X to mode Y , with Y different from X . Thus, e.g., a counter can go from no-change to increasing, or from increasing to decreasing, etc. We note that since the machine executes its program sequentially (one instruction at a time), no two counters can make a mode-change at the same time. Assume there are k counters. At any time during the computation, the modes of the counters can be represented by a mode-vector $Q = \langle m_1, \dots, m_k \rangle$, where m_i is the mode of the i -th counter, for $1 \leq i \leq k$. There are only a finite number (3^k) of such vectors. The behavior of the counters during an accepting computation (which, by definition, is a halting computation) can be represented by a sequence: $Q_1 N_1 Q_2 N_2 \dots Q_t N_t$ where: (1) The Q_i 's are mode-vectors, (2) Each N_i represents the (possibly empty) period when no counter changes mode, (3) For each $1 \leq i \leq t-1$, Q_{i+1} differs from Q_i in exactly one component (i.e., exactly one counter changes mode), and (4) The starting mode-vector $Q_1 = \langle \text{no-change}, \dots, \text{no-change} \rangle$.

Thus, we can divide the computation into phases, where in each phase, no counter changes mode. Now since the machine is strongly reversal-bounded, t is upper-bounded by some fixed number.

Call the sequence $\langle Q_1, \dots, Q_t \rangle$ a Q -vector. (Note that since each Q_i is a k -tuple, the Q -vector has $k \times t$ components.) Since t is upper-bounded by some fixed number, there are only a finite number of such Q -vectors.

We now prove Theorem 8. Let M be a nondeterministic finite-crossing strongly reversal-bounded multicounter machine that uses atomic linear-relation predicates. We describe the construction of an equivalent nondeterministic finite-crossing strongly reversal-bounded multicounter machine M' (which may have more counters than M) that uses only the standard instructions.

The construction of M' is an induction on the number of atomic linear relations occurring in the program of M . Consider a specific instruction, say labeled s (i.e. state s) of the form “**if** L **then goto** p **else goto** q ” in the program of M , where L is a linear relation. W.l.o.g, we assume that L is an atomic linear relation. We will construct an equivalent strongly reversal-bounded machine M' without this instruction (i.e., M' has one less atomic linear relation). Note that M' cannot simply implement this conditional using the standard instructions since the conditional will require a finite number of reversals on the counters of M' . If this conditional is executed by M an unbounded number of times during the computation, the counters of M' will not be reversal-bounded.

The basic idea in the construction of M' is as follows:

1. M' stores in its states the atomic linear relation L .
2. M' first guesses and stores in its states a Q -vector $\langle Q_1, \dots, Q_t \rangle$.
3. M' simulates M by phases, where each phase starts with mode-vector Q_i and ends with mode-vector Q_{i+1} . We assume that M keeps track of the values of the counters of M during the computation and, in particular, has available in its counters the values of the counters of M at the start and end of each phase. We also assume that M' keeps track of the state changes of M . In the simulation, M' does not use instruction “ $s : \text{if } L \text{ then goto } p \text{ else goto } q$.”

We give the details of simulating a phase starting at Q_i and ending at Q_{i+1} :

1. M' first checks, using the values of the counters involved in the conditional “ $s : \text{if } L \text{ then goto } p \text{ else goto } q$ ” whether L is true or whether it is false at the beginning of the phase.
2. Consider the case when L is true (the case when C is false is symmetric). There are two subcases:

Subcase 1: Throughout the phase, L remains true. Since L is an atomic linear relation, it is convex. It follows that L is true throughout the phase if and only if it is true at the start and at the end of the phase.

Subcase 2: During the computation, L became false. Again since L is convex, when it turns false it will remain false until the end of the phase. Moreover, the time when L becomes false is unique (i.e., it only occurs once in the entire phase).

So, to simulate a phase, M' guesses one of the two subcases above. Suppose M' guesses Subcase 1. Then it simulates M' faithfully using the instruction “**goto** p ” in place of “ $s : \text{if } L \text{ then goto } p \text{ else goto } q$ ” until the end of the phase. At the end of the phase it verifies that L is still true.

Suppose M' guesses Subcase 2. Then it simulates M' faithfully. But, in addition, M' guesses the last time, u , the conditional instruction will be executed by M with value true (meaning the conditional instruction becomes false at the $(u + 1)$ -st time it is executed by M).

Up to time u , M' uses the instruction “**goto** p ”.

After time u , M' uses the instruction “**goto** q ”.

M' also verifies that at time u , L is indeed true, and it is false at time $u+1$.

It follows from the description above that we can remove the instruction “ s : **if** L **then goto** p **else goto** q ” and M' is still strongly-reversal bounded. We can iterate the process to remove all atomic linear-relation conditionals.

4.3 Allowing Parameterized Constants

We can further generalize our model by allowing parameterized constants in the linear relations. So for example, we can allow instructions like

s : **if** $3x - 5y - 2D_1 + 9D_2 < 12$ **then goto** p **else goto** q

where D_1 and D_2 represent parameterized constants whose domain is the set of all integers $(+, -, 0)$. We can specify the values of these parameters at the start of the computation by including them on the input tape. Thus, the input to the machine with k parameterized constants will have the form “ $\#d_1\% \dots \%d_k\%w\#$ ”, where $d_1, \dots, d_k$ are integers $(+, -, 0)$ that the parameterized constants $D_1, \dots, D_k$ assume for this run, and $\%$ is a separator. We assume that the d_i ’s are represented in unary along with their signs.

Theorem 9. The emptiness problem is decidable for nondeterministic finite-crossing strongly reversal-bounded multicounter machines using linear-relation conditionals on the counters and parameterized constants.

4.4 Allowing One Unrestricted Counter

We can allow one of the counters to be unrestricted (i.e., not reversal-bounded) provided the input is one-way:

Theorem 10. The emptiness problem is decidable for nondeterministic one-way machines with one unrestricted counter and several strongly reversal-bounded counters using linear-relation conditionals on the reversal-bounded counters and parameterized constants.

4.5 Restricted Linear Relations

Because of Theorem 6, none of Theorems 8–10 holds when the machines are reversal-bounded but not strongly reversal-bounded. However, suppose we require that in every linear relation L , every atomic linear relation in L involves only the parameterized constants and at most one counter so, e.g., $4D_1 + 9D_2 < 7$ and $5x - 4D_1 + 9D_2 < 7$ are fine, but $5x + 2y - 4D_1 + 9D_2 < 7$ is not (where x and y are counters, and D_1 and D_2 are parameterized constants). Call such a relation L a *restricted linear relation*. Then the results of Theorems 8–10 hold for reversal-bounded machines (which are not necessarily strongly reversal-bounded) that use such restricted linear relations.

Finally, we state that Theorems 4 and 5 can be generalized:

Theorem 11. The infiniteness, disjointness, containment, and equivalence problems are decidable for various generalizations introduced in this section (with containment and equivalence holding only for deterministic machines).

5 Reachability, Safety, and Invariance

The results of the previous section can be used to analyze verification problems (such as reachability, safety, and invariance) in infinite-state transition systems that can be modeled by multicounter machines. Decidability of reachability is of importance in the areas of model checking, verification, and testing [7,3,15]. In these areas, a machine is used as a system specification rather than a language recognizer, the interest being more in the behaviors that the machine generates. Thus, in this section, unless otherwise specified, the machines have *no* input tape.

For notational convenience, we restrict our attention to machines whose counters can only store nonnegative integers. The results easily extend to the case when the counters can be negative.

Let M be a nondeterministic reversal-bounded k -counter machine with state set $\{1, 2, \dots, s\}$ for some s . Each counter can be incremented by integer constants $(+, -, 0)$ and can be tested if $<, >, =$ to integer constants. Let $(j, v_1, \dots, v_k)$ denote the configuration of M when it is in state j , and counter i has value v_i for $i = 1, 2, \dots, k$. Thus, the set of all possible configurations is a subset of $\mathbb{N}^{k+1}$.

Given M , let $R(M) = \{(j, v_1, \dots, v_k, j', v'_1, \dots, v'_k) \mid \text{configuration } (j, v_1, \dots, v_k) \text{ can reach configuration } (j', v'_1, \dots, v'_k) \text{ in 0 or more moves}\}$. $R(M)$, which is a subset of $\mathbb{N}^{2k+2}$, is called the *binary reachability* set of M . For a set S of configurations, define $post^*_M(S)$ to be the set of all successors of configurations in S , i.e., $post^*_M(S) = \{\alpha \mid \alpha \text{ can be reached from some configuration in } S \text{ in 0 or more moves}\}$. Similarly, define $pre^*_M(S) = \{\alpha \mid \alpha \text{ can reach some configuration in } S \text{ in 0 or more moves}\}$. $post^*_M(S)$ and $pre^*_M(S)$ are called the *forward* and *backward reachability* of M with respect to S , respectively.

Note that configuration $(j, v_1, \dots, v_k)$ in $\mathbb{N}^{k+1}$ can be represented as a string $j\%v_1\% \dots \%v_k$, where $j, v_1, \dots, v_k$ are represented in unary (separated by $\%$). Thus, $pre^*_M(S)$ and $post^*_M(S)$ can be viewed as languages (e.g., regular, context-free, etc.). Similarly, $R(M)$ can be viewed as a language.

When we say that a subset S of $\mathbb{N}^n$ is accepted by a multicounter machine M , we mean that, M when started in its initial state with its first n counters (M can have more than n counters) set to an n -tuple accepts (i.e., enters an accepting state) if and only if the n -tuple is in S . Note that this is equivalent to equipping the machine with an input tape that contains the unary encoding of the n -tuple.

We state the following characterization theorem without proof.

Theorem 12. Let M be a nondeterministic reversal-bounded k -counter machine and S a subset of $\mathbb{N}^{k+1}$.

1. $R(M)$ is definable by a Presburger formula.
2. S is definable by a Presburger formula if and only if $post^*_M(S)$ ($pre^*_M(S)$) is definable by a Presburger formula.
3. $post^*_M(S)$ ($pre^*_M(S)$) can be accepted by a deterministic reversal-bounded multicounter machine.
4. 1–3 still hold even if one of the counters is unrestricted.

Next, we consider strongly reversal-bounded multicounter machines with linear-relation conditionals on the counters and parameterized constants. For these machines, the configuration is now a tuple $(j, v_1, \dots, v_k, d_1, \dots, d_m)$, where $d_1, \dots, d_m$ represent the values of the parameterized constants. Then the following theorem follows from the results of the previous section:

Theorem 13. The statements in Theorem 12 hold for:

1. M a nondeterministic strongly reversal-bounded k -counter machine using linear relation conditions on the counters and parameterized constants.
2. M a nondeterministic reversal-bounded k -counter machine using restricted linear relation conditions on the counters and parameterized constants.

The results are valid even if one of the counters is unrestricted as long as this counter is not involved in the linear relation conditionals.

Theorem 14. Consider deterministic machines with one unrestricted counter U , k reversal-bounded counters, and a finite number of parameterized constants. In addition to the standard instructions, the unrestricted counter can be tested for “ $U = D$?” where D represents a parameterized constant. Then

1. The emptiness problem (i.e., deciding given a machine M whether there exists an assignment of values to the parameterized constants which will cause M to accept) is undecidable, even when restricted to $k = 2$.
2. The emptiness problem is decidable when $k = 1$.
3. The emptiness problem is decidable for any k , provided there is only *one* parameterized constant.

The problems of *safety* and *invariance* are of importance in the area of verification. The following theorem follows from Theorems 12 and 13.

Theorem 15. In the following, M is a nondeterministic reversal-bounded multicounter machine and S and T are two sets of configurations accepted by nondeterministic reversal-bounded multicounter machines:

1. It is decidable to determine whether there is no configuration in S that can reach a configuration in T . Thus, safety is decidable with respect to a bad set of configurations T .
2. It is decidable to determine whether every configuration in S can only reach configurations in T . Thus, invariance is decidable with respect to a good set T .

6 Conclusions

We introduced several generalizations of reversal-bounded multicounter machines and investigated their decision problems. We then used the decidable properties to analyze verification problems such as (binary, forward, backward) reachability and safety. In practice, many infinite-state transition systems can be modeled by multicounter machines. In order to debug a safety property, a multicounter machine can be effectively made

reversal-bounded by giving a bound on reversals. This is a new approach for analyzing safety properties where the general reachability problem is known to be undecidable. Other approaches use semi-decision algorithms that are not guaranteed to terminate [15] or to look at restricted classes of systems where reachability is decidable [3].

It may seem that strong reversal-boundedness restricts the behavior of a counter too much, since changing from a strictly increasing (or decreasing) mode to a no-change mode counts as a reversal. However, if the counters behave like clocks which either increase with rate 1 or reset to 0 as in timed automata [1], strong reversal-boundedness is equivalent to reversal-boundedness. Using this observation and the results in this paper, we are able to show a number of results on clocked systems with parameterized durations [4], binary reachability characterization of discrete timed pushdown automata [5], and reachability of past machines [6].

Acknowledgment. The work by Ibarra and Su was supported in part by NSF grant IRI-9700370; the work by Bultan was supported in part by NSF grant CCR-9970976; the work by Dang and Kemmerer was supported in part by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Material Command, USAF, under agreement number F30602-97-1-0207.

References

1. R. Alur and D. Dill. "A theory of timed automata," *Theo. Comp. Sci.*, 126(2):183-235, 1994.
2. B. Baker and R. Book. "Reversal-bounded multipushdown machines," *J.C.S.S.*, 8:315-332, 1974.
3. H. Comon and Y. Jurski. "Multiple counters automata, safety analysis and Presburger arithmetic," *Proc. Int. Conf. on Computer Aided Verification*, pp. 268-279, 1998.
4. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su. "Decidable approximations on discrete clock machines with parameterized durations," in preparation.
5. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su. "Binary reachability analysis of discrete pushdown timed automata," to appear in CAV 2000.
6. Z. Dang, O. H. Ibarra, T. Bultan, R. A. Kemmerer, and J. Su. "Past machines," in preparation.
7. J. Esparza. "Decidability of model checking for infinite-state concurrent systems," *Acta Informatica*, 34(2):85-107, 1997.
8. E. M. Gurari and O. H. Ibarra. "Simple counter machines and number-theoretic problems," *J.C.S.S.*, 19:145-162, 1979.
9. O. H. Ibarra. "Reversal-bounded multicounter machines and their decision problems," *J. ACM*, 25:116-133, 1978.
10. O. H. Ibarra, T. Jiang, N. Tran, and H. Wang. "New decidability results concerning two-way counter machines," *SIAM J. Comput.*, 24(1):123-137, 1995.
11. Y. Matijasevic. "Enumerable sets are Diophantine," *Soviet Math. Dokl*, 11:354-357, 1970.
12. M. Minsky. "Recursive unsolvability of Post's problem of Tag and other topics in the theory of Turing machines." *Ann. of Math.*, 74:437-455, 1961.
13. R. Parikh. "On context-free languages," *J. ACM*, 13:570-581, 1966.
14. L. G. Valiant and M. S. Paterson. "Deterministic one-counter automata," *J.C.S.S.*, 10:340-350, 1975.
15. P. Wolper and B. Boigelot. "Verifying systems with infinite but regular state spaces," *Proc. 10th Int. Conf. on Computer Aided Verification*, pp. 88-97, 1998.

A Family of NFA's Which Need $2^n - \alpha$ Deterministic States

Kazuo Iwama¹, Akihiro Matsuura¹, and Mike Paterson²

¹ School of Informatics, Kyoto University

² Department of Computer Science, University of Warwick

E-mail: {iwama, matsu}@kuis.kyoto-u.ac.jp, msp@dcs.warwick.ac.uk

Abstract. We show that for all integers $n \geq 7$ and α , such that $5 \leq \alpha \leq 2n - 2$ and satisfying some coprimality conditions, there exists a minimum n -state nondeterministic finite automaton that is equivalent to a minimum deterministic finite automaton with exactly $2^n - \alpha$ states.

1 Introduction

Finite automata theory is obviously a popular first step to theoretical computer science, through which students learn several basic notions of computation models. Nondeterminism might be the most important one among those notions. The subset construction [1], which shows that any nondeterministic finite automaton (NFA) can be simulated by a deterministic finite automaton (DFA), is probably one of the oldest, non-trivial theorems in this field. This theorem is often stated as above, i.e., “NFA’s are no stronger than DFA’s”, but we have to be careful since the simulation is only possible “by increasing the number of states”. Since the number of states is the principal complexity measure for finite automata, the extent to which NFA’s are more efficient than DFA’s is an important feature and provides the basis for the same relationship in stronger models.

It is known [2], [3] that there is an NFA of n states which needs 2^n states to be simulated by a DFA. Thus some NFA’s are exponentially more efficient than DFA’s in terms of the number of states. Of course, however, this is not always true; for example, the DFA which counts the number of 1’s modulo k needs k states and equivalent NFA’s need the *same* number of states. So, nondeterminism works very well for some kind of languages and does not for others. Thus it is of interest to ask which kinds of language belong to the first category and which to the second.

It is hard to give a general answer to this problem. However, one simple and concrete question regarding this problem is the following: *For a positive integer n , is there an integer Z , $n < Z < 2^n$, such that no DFA of Z states can be simulated by any NFA of n states?* Such a number Z or the one that satisfies the above question for all n can be regarded as a “magic number” for which nondeterminism is especially weak. It turns out that to answer this question,

we have only to consider $2^{n-1} \leq Z < 2^n$. Furthermore, 2^{n-1} cannot be such a magic number [4]. If there are no such magic numbers at all, which seems more likely to us, that means that for any integer $0 \leq \alpha \leq 2^{n-1} - 1$, there is an NFA of n states which needs $2^n - \alpha$ deterministic states.

This question was first considered by Iwama, Kambayashi and Takaki [5]. They show that if an integer α can be expressed as 2^k or $2^k + 1$ for some integer $0 \leq k \leq n/2 - 2$, then there is an NFA of n states which needs $2^n - \alpha$ deterministic states, i.e., such $2^n - \alpha$ cannot be a magic number in the above sense. In this paper, we give a somewhat (but not yet completely) general answer. Namely, for all integers $n \geq 7$ and α , such that $5 \leq \alpha \leq 2n - 2$ and with some coprimality condition, $2^n - \alpha$ cannot be a magic number. Furthermore, we show that $2^n - 6$ cannot be a magic number, unconditionally. Note that $2^n - 6$ is the largest number which cannot be expressed as 2^k or $2^k + 1$, and so was left open in [5].

2 Main Results

A finite automaton M is determined by the following five items: a finite set of states; a finite set of input symbols Σ , which is always $\{0, 1\}$ in this paper; an initial state; a set of accepting states; and a state transition function δ .

Our main task in this paper is (i) to give an NFA M , (ii) to find the equivalent DFA, (iii) to analyze the number of states in the DFA which can be reached from its initial state, and finally (iv) to show that all such states are inequivalent. For (ii), we use the so-called subset construction [1], i.e., each state of the DFA is given as a subset of M 's states and the resulting DFA is written as $D(M)$. To avoid confusion, a state of $D(M)$ will be called an *f-state* (f stands for family). We always use δ for the state transition function of $D(M)$. Two f-states Q_1 and Q_2 are *equivalent* if for all $x \in \Sigma^*$, $\delta(Q_1, x) \in F$ iff $\delta(Q_2, x) \in F$, where F is the set of accepting states in $D(M)$. Suppose on the other hand that we wish to show that two f-states Q_1 and Q_2 are not equivalent. Then, what we should do is (i) to show that $Q_1 \in F$ and $Q_2 \notin F$ (or vice versa), or (ii) to find a string $x \in \Sigma^*$ such that $\delta(Q_1, x)$ and $\delta(Q_2, x)$ are already known to be inequivalent. For an NFA M of n states, $\Delta(M)$ denotes the number of states of a minimum DFA which is equivalent to M . It is well known [1] that a DFA is minimum if all of its states can be reached from the initial state and no two states are equivalent. Now we are ready to give our results.

Theorem 1. *Let n and α be any integers such that $5 \leq \alpha \leq n-1$, $6 \leq \alpha \leq n$, or $9 \leq \alpha \leq 2n - 2$ and such that n is relatively prime with $\alpha - 1$, $\alpha - 2$, or $\lceil \alpha/2 \rceil - 1$, respectively. Then there exists a minimum n -state NFA whose equivalent minimum DFA has $2^n - \alpha$ states.*

Corollary 1. *For all integers $n \geq 7$ and α , such that $5 \leq \alpha \leq 2n - 2$ and satisfying the coprimality condition in Theorem 1, there exists an n -state NFA whose equivalent minimum DFA has $2^n - \alpha$ states.*

Note that for $\alpha \leq 5$, it was shown in [5] that there exists an n -state NFA M such that $\Delta(M) = 2^n - \alpha$ for $n \geq 8$.

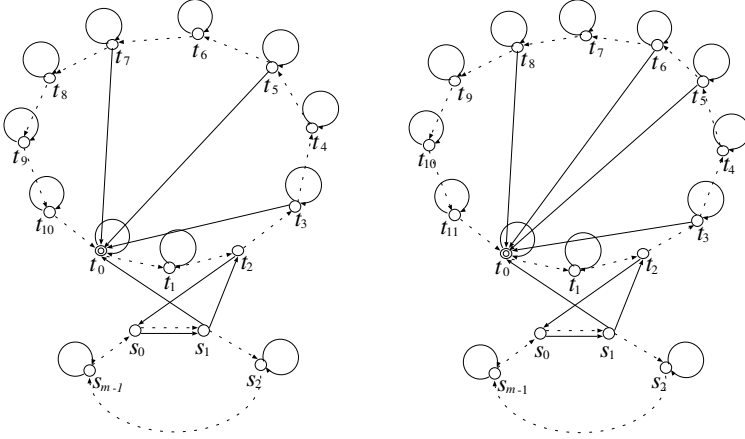


Fig. 1. (a) M_1 when k is odd ($k = 11$) and (b) M_1 when k is even ($k = 12$)

The next theorem is less general, but does not need the coprimality condition. Recall that $2^n - 6$ was the first unsettled number in [5].

Theorem 2. *For any $n \geq 5$, there exists an n -state NFA whose equivalent minimum DFA has $2^n - 6$ states.*

3 Proof of Theorem 1

For ease of explanation, we introduce the parameter k that represents $\alpha - 1$, $\alpha - 2$, or $\lceil \alpha/2 \rceil - 1$, corresponding to the three cases in the hypothesis, and we suppose that k and n have no common divisor. Let m denote $n - k$, i.e., $n = k + m$, then k and m also have no common divisor. In this section, we first give an NFA M_1 whose equivalent minimum DFA has $2^n - (k + 1)$ states. Then we give five lemmas which give the number of f-states in $D(M_1)$ and claim that no two f-states are equivalent. M_1 is illustrated in Fig. 1. Its state set is the union of $T = \{t_0, t_1, \dots, t_{k-1}\}$ and $S = \{s_0, s_1, \dots, s_{m-1}\}$. Its initial state is t_0 . Note that $|T \cup S| = k + n = m$. A set in T (in S , resp.) is called a T -state (S -state, resp.). State transitions on reading 0 (denoted by dotted arrows in Fig. 1) are cyclic, i.e., $t_0 \xrightarrow{0} t_1, t_1 \xrightarrow{0} t_2, \dots, t_{k-1} \xrightarrow{0} t_0$, and $s_0 \xrightarrow{0} s_1, \dots, s_{m-1} \xrightarrow{0} s_0$. Transitions on reading 1 are as follows: For all i , $0 \leq i \leq k-1$ excepting $i = 2$, there are self-loops as $t_i \xrightarrow{1} t_i$. Similarly, $s_j \xrightarrow{1} s_j$ for $2 \leq j \leq m-1$. In addition, there are transitions of the form $t_i \xrightarrow{1} t_0$ where $i = 3, 5, \dots, k-4$ when k is odd. When k is even, these transitions are defined for $i = 3, 5, \dots, 2r-3, 2r-1, 2r, 2r+2, \dots, k-4$ where $r = \lceil k/4 \rceil$. The remaining four transitions are $s_0 \xrightarrow{1} s_1, s_1 \xrightarrow{1} t_0, s_1 \xrightarrow{1} t_2$, and $t_2 \xrightarrow{1} s_0$.

For any f-state P , $P \cap T$ is called the T -portion of P and denoted by P_T . Similarly, $P \cap S$ is called the S -portion of P and denoted by P_S . The size of P ,

$|P|$, is the number of M_1 's states included in P . The transition on T (or S) that occurs on reading 0 is called a 0 -shift. The index of a state is considered to be modulo k ; namely, the 0-shift of t_i is always written as t_{i+1} .

The first lemma deals with exceptional f-states P such that $|P_S| = 0$ and $|P_T| = 0, 1$, and 2. We say that an f-state Q_1 is *reachable from* an f-state Q_2 if there is a string $x \in \Sigma^*$ such that $Q_1 = \delta(Q_2, x)$. If $Q_1 = \delta(\{t_0\}, x)$, then we simply say that Q_1 is *reachable*. Q_1 is said to be *unreachable* if it is not reachable.

Lemma 1. *For an f-state P such that $|P_S| = 0$ and $0 \leq |P_T| \leq 2$, the following statements hold. (1) When $|P_T| = 0$, there is only one f-state, ϕ (the empty set), and this is unreachable. (2) When $|P_T| = 1$, P is reachable. (3) When $|P_T| = 2$, P is reachable unless P consists of two neighboring states of T , that is, $P = \{t_i, t_{i+1}\}$ ($i = 0, 1, \dots, k-1$).*

Note that there are $k+1$ unreachable f-states given in this lemma. The remaining $2^n - (k+1)$ f-states are all reachable, which is shown by the following four lemmas depending on (i) whether $|P_S| = 0$ or $|P_S| > 0$ and (ii) whether or not P_T contains two states of distance two, i.e., t_i and t_{i+2} . Distance-two states are important since the transition $s_1 \xrightarrow{1} \{t_0, t_2\}$ plays a special role in M_1 .

Lemma 2. *For an f-state P such that $|P_S| = 0$ and $|P_T| \geq 3$, if P contains a pair of states, t_i and t_{i+2} for some $i = 0, 1, \dots, k-1$ (P may include t_{i+1} as well), P is reachable from some f-state Q such that $(|Q_S|, |Q_T|) = (1, |P_T| - 2)$.*

Lemma 3. *For an f-state P such that $|P_S| = 0$ and $|P_T| \geq 3$, if P does not contain a pair of states, t_i and t_{i+2} for any $i = 0, 1, \dots, k-1$, P is reachable from some f-state Q such that $(|Q_S|, |Q_T|) = (0, |P_T| - 1)$. Furthermore, if $|P_T| = 3$, $Q_T \neq \{t_i, t_{i+1}\}$, i.e., the two states of Q_T are not neighboring.*

Lemma 4. *For an f-state P such that $|P_S| \geq 1$, if P contains a pair of states t_i and t_{i+2} for some $i = 0, 1, \dots, k-1$, P is reachable from some f-state Q such that $(|Q_S|, |Q_T|) = (|P_S|, |P_T| - 1)$.*

Lemma 5. *For an f-state P such that $|P_S| \geq 1$, if P does not contain a pair of states t_i and t_{i+2} for any $i = 0, 1, \dots, k-1$, P is reachable from some f-state Q such that $(|Q_S|, |Q_T|) = (|P_S| - 1, |P_T| + 1)$. Furthermore, when $(|P_S|, |P_T|) = (1, 1)$, $Q_T \neq \{t_i, t_{i+1}\}$, i.e., the two states of Q_T are not neighboring.*

See Table 1, which summarizes Lemmas 1 to 5 and also summarizes our induction scheme to claim how each f-state is reachable for an odd k . The leftmost three entries (1, 0, and k) in its first row show the numbers of unreachable f-states described in Lemma 1. Dotted arrows show the reachability described in Lemmas 3 and 5. Solid arrows show the reachability given in Lemmas 2 and 4. For example, the entry for $(|P_S|, |P_T|) = (0, 4)$ receives a dotted arrow from $(|P_S|, |P_T|) = (0, 3)$ and a solid arrow from $(|P_S|, |P_T|) = (1, 2)$. Two dotted arrows from $(|P_S|, |P_T|) = (0, 2)$ need special care since this entry includes unreachable f-states, or we have to show that those reachabilities do not start from such unreachable states. Also, one should notice that there are no dotted arrows to any P such that $|P_T| \geq (k+1)/2$. The reason is that if

Table 1. Number of unreachable f-states when k is odd

$ P_S \backslash P_T $	0	1	2	3	4	$\dots$	$(k-1)/2$	$(k+1)/2$	$(k+3)/2$	$\dots$	$k-1$	k
0	1	0	k	0	0	$\dots$	0	0	0	$\dots$	0	0
1	0	0	0	0	0	$\dots$	0	0	0	$\dots$	0	0
2	0	0	0	0	0	$\dots$	0	0	0	$\dots$	0	0
3	0	0	0	0	0	$\dots$	0	0	0	$\dots$	0	0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$m-1$	0	0	0	0	0	$\dots$	0	0	0	$\dots$	0	0
m	0	0	0	0	0	$\dots$	0	0	0	$\dots$	0	0

$|P_T| \geq (k+1)/2$, then P_T must include a pair of distance-two states. Altogether, each f-state P such that $(|P_S|, |P_T|) = (1, 0)$ is reachable from Q such that $(|Q_S|, |Q_T|) = (0, 1)$, and such a Q is reachable from $\{t_0\}$ by Lemma 1. All the other f-states are reachable by traversing solid and dotted arrows starting from $(|P_S|, |P_T|) = (0, 2)$, and the latter f-states are reachable by Lemma 1.

3.1 Proof of Lemma 1

ϕ is obviously unreachable since every state in M_1 has non-empty transitions on reading 0 and 1. When $|P_T| = 1$, P can be written as $\{t_i\}$, which is reachable from $\{t_0\}$, the initial f-state, by 0-shifts.

We now consider the case $(|P_S|, |P_T|) = (0, 2)$, which is divided into two cases according to whether or not P contains a neighboring pair of states in T . The argument is a little different for odd and even k 's. In the following, we only consider the odd case. First, suppose that $P = \{t_0, t_i\}$, where $i \neq 1, k-1$, namely the two states of P are not neighboring. When $i = 3, 5, 7, \dots, k-4$, we can use the following transitions:

$$\{t_0\} \xrightarrow{0^i} \{t_i\} \xrightarrow{1} \{t_0, t_i\}.$$

When $i = 2$ and $k-2$, we can follow

$$\{t_0\} \xrightarrow{0^2} \{t_2\} \xrightarrow{1} \{s_0\} \xrightarrow{1} \{s_1\} \xrightarrow{1} \{t_0, t_2\} \xrightarrow{0^{k-2}} \{t_0, t_{k-2}\}.$$

When $i = 4, 6, 8, \dots, k-3$, we can follow

$$\{t_0\} \xrightarrow{0^{k-i}} \{t_{k-i}\} \xrightarrow{1} \{t_0, t_{k-i}\} \xrightarrow{0^i} \{t_0, t_i\}.$$

Thus $P = \{t_0, t_i\}$ is reachable unless $i = 1$ or $i = k-1$. All other non-neighboring f-states are reachable from $\{t_0, t_i\}$ by 0-shifts.

As for a neighboring pair of states such as $\{t_i, t_{i+1}\}$, this is shown to be unreachable as follows. First of all, one can see that if we do not use the transition from t_2 to s_0 , we can never reach $\{t_i, t_{i+1}\}$, for the following reasons. We start from $\{t_0\}$. Then, if we use only transitions between T -states, which we call T -transitions, then the size $|P|$ of the current f-state P monotonically increases. Hence, consider the moment when $|P|$ changes from one to two. The transition used at this moment must be $t_i \xrightarrow{1} t_i$ and $t_i \xrightarrow{1} t_0$. It then follows that P cannot be neighboring since we have no such transitions from t_1 or t_{k-1} . It is easy to see that such a P cannot later change to a pair of neighboring states while $|P| = 2$. Thus there must be an f-state which includes some S -state on the way from $\{t_0\}$ to $\{t_i, t_{i+1}\}$ (if any). Let K be the last f-state including S -states. Then, symbol 1 must be read on state K , since otherwise $\delta(K, 0)$ still contains both S - and T -states. Furthermore, K never contains s_1 , since otherwise $\delta(K, 1)$ includes $\{t_0, t_2\}$, which cannot change to a pair of neighboring states by using T -transitions. Hence, K must only contain some S -state other than s_1 , but this contradicts our assumption for K . $\square$

3.2 Proof of Lemma 2

Suppose that $\{t_i, t_{i+2}\} \subset P$. Obviously, P is reachable from some P' such that $\{t_0, t_2\} \subset P'$. Now we can see that $Q = (P' \setminus \{t_0, t_2\}) \cup \{s_1\} \xrightarrow{1} P'$, where $P' \setminus \{t_0, t_2\}$ means that $\{t_0, t_2\}$ is removed from P' . Thus Q satisfies the condition of the lemma, i.e., $(|Q_S|, |Q_T|) = (1, |P_T| - 2)$. $\square$

3.3 Proof of Lemma 3

Now P does not include any $\{t_i, t_{i+2}\}$. Without loss of generality, we can assume that P contains t_0 (otherwise, P is reachable from such an f-state by 0-shifts). Hence, let $P = \{t_0, t_{p_1}, t_{p_2}, \dots, t_{p_{r-1}}\}$, where $|P| \geq 3$ and $p_1 = 1$ or $p_1 \geq 3$ since there is no pair of distance-two states. The proof differs slightly according to whether k is odd or even (recall that our machine M_1 is different for odd and even k 's). We first prove the lemma for an odd k and the difference in the even case will be briefly given. There are several cases to be considered.

(Case 1) $p_1 = 1$, namely, $P = \{t_0, t_1, t_{p_2}, \dots, t_{p_{r-1}}\}$. This case is further divided into two subcases according to whether p_2 is odd or even.

(Case 1-1) p_2 is odd. By the assumption of Lemma 3, $4 \leq p_2 \leq k-3$, and since p_2 was assumed to be odd, $5 \leq p_2 \leq k-4$. Therefore, we can use the transition $t_{p_2} \xrightarrow{1} \{t_0, t_{p_2}\}$, namely:

$$Q = P \setminus \{t_0\} = \{t_1, t_{p_2}, \dots, t_{p_{r-1}}\} \xrightarrow{1} \{t_0, t_1, t_{p_2}, \dots, t_{p_{r-1}}\} = P.$$

Note that Q satisfies the condition of the lemma, i.e., $(|Q_S|, |Q_T|) = (0, |P_T| - 1)$. It should be noted that for $r = 3$, $Q = \{t_1, t_{p_2}\}$ is known to be reachable by Lemma 1 because t_1 and t_{p_2} are not neighboring.

(Case 1-2) p_2 is even. Since $4 \leq p_2 \leq k-3$ and p_2-1 is odd, $3 \leq p_2-1 \leq k-4$. Therefore, there is a transition $t_{p_2-1} \xrightarrow{1} \{t_0, t_{p_2-1}\}$. Let $P' = \delta(P, 0^{k-1}) = \{t_{k-1}, t_0, t_{p_2-1}, t_{p_3-1}, \dots, t_{p_{r-1}-1}\}$. Then, we can use the following sequence of transitions:

$$Q = P' \setminus \{t_0\} \xrightarrow{1} P' \xrightarrow{0} P.$$

For $r = 3$, $Q = \{t_{k-1}, t_{p_2-1}\}$ is not neighboring again.

(Case 2) $p_1 \geq 3$. We can assume that t_{p_1} and t_{p_2} are not neighboring, since otherwise we can apply the argument of Case 1. Also note that $p_1 \leq k-3$ (otherwise P , $|P| \geq 3$, clearly includes a pair of distance-two states).

(Case 2-1) p_1 is odd. We have the following direct transition to P using the transitions $t_{p_1} \xrightarrow{1} \{t_0, t_{p_1}\}$ in M_1 :

$$P \setminus \{t_0\} = \{t_{p_1}, t_{p_2}, \dots, t_{p_{r-1}}\} \xrightarrow{0} P.$$

(Case 2-2) p_1 is even. Since $4 \leq p_1 \leq k-3$ and $k-p_1$ is odd, $3 \leq k-p_1 \leq k-4$. This time we use $t_{k-p_1} \xrightarrow{1} \{t_0, t_{k-p_1}\}$. Let $P' = \delta(P, 0^{k-p_1}) = \{t_{k-p_1}, t_0, t_{p_2-p_1}, \dots, t_{p_{r-1}-p_1}\}$. Then, we obtain the following sequence of transitions:

$$P' \setminus \{t_0\} \xrightarrow{1} P' \xrightarrow{0^{p_1}} P.$$

Thus, P is reachable from $Q = P' \setminus \{t_0\}$. Again for $r = 3$, $Q = \{t_{k-p_1}, t_{p_2-p_1}\}$ is not neighboring and is known to be reachable by Lemma 1. Consequently, it has been shown that in all cases, there is a transition of the form $Q \longrightarrow P$, where Q satisfies $(|Q_S|, |Q_T|) = (0, |P_T| - 1)$. $\square$

For an even k , it is divided into three cases: $p_1 = 1$, $3 \leq p_1 \leq k/2 - 1$, and $p_1 \geq k/2$. In each case, the reachability of the f-states are shown in the similar way to the odd case, where the transitions of type $t_i \xrightarrow{1} \{t_0, t_i\}$ play the essential role again.

3.4 Proof of Lemma 4

Recall that $\{t_i, t_{i+2}\} \subset P$ and $|P_S| \geq 1$. We consider two cases, one for $|P_S| \leq m-1$ and the other for $|P_S| = m$.

(Case 1) $|P_S| \leq m-1$. Since k and n have no common divisor and since $P_S \neq S$, there is an f-state P' such that (i) P is reachable from P' , (ii) $\{t_0, t_2\} \subset P'$, and (iii) $s_0 \in P'$ and $s_1 \notin P'$. Let $P'_1 = P'_T \setminus \{t_0, t_2\}$ and $P'_2 = P'_S \setminus \{s_0\}$. Then, one can use the following transition:

$$Q = (P'_1 \cup \{t_2\}) \cup (P'_2 \cup \{s_1\}) \xrightarrow{1} (P'_1 \cup \{s_0\}) \cup (P'_2 \cup \{t_0, t_2\}) = P',$$

since P'_1 and P'_2 do not change on reading 1, $\{t_2\} \xrightarrow{1} \{s_0\}$, and $\{s_1\} \xrightarrow{1} \{t_0, t_2\}$. Note that $(|Q_S|, |Q_T|) = (|P_S|, |P_T| - 1)$ and the lemma follows.

(Case 2) $|P_S| = m$. Namely, $P_S = S$. Similarly to Case 1, there is an f-state P' such that (i) P is reachable from P' , (ii) $\{t_0, t_2\} \subset P'$, and (iii) $P'_S = P_S = S$.

Let $P'_1 = P'_T \setminus \{t_0, t_2\}$ and $P'_2 = P_S \setminus \{s_0, s_1\}$. Then one can use the following transition:

$$Q = (P'_1 \cup \{t_2\}) \cup (P'_2 \cup \{s_0, s_1\}) \xrightarrow{1} (P'_1 \cup \{s_0\}) \cup (P'_2 \cup \{s_1, t_0, t_2\}) = P',$$

where $(|Q_S|, |Q_T|) = (|P_S|, |P_T| - 1)$. $\square$

3.5 Proof of Lemma 5

Suppose that P does not include any $\{t_i, t_{i+2}\}$. We consider two cases similarly to Section 3.4.

(Case 1) $|P_S| \leq m - 1$. As before, there is an f-state P' such that (i) P is reachable from P' , (ii) $t_0 \in P'$ and $t_2 \notin P'$, and (iii) $s_0 \in P'$ and $s_1 \notin P'$. Let $P'_1 = P'_T \setminus \{t_0\}$ and $P'_2 = P'_S \setminus \{s_0\}$. Then, one can use the following transition:

$$Q = (P'_1 \cup \{t_0, t_2\}) \cup P'_2 \xrightarrow{1} (P'_1 \cup \{t_0, s_0\}) \cup P'_2 = P',$$

where $(|Q_S|, |Q_T|) = (|P_S| - 1, |P_T| + 1)$.

(Case 2) $|P_S| = m$. In this case, there is an f-state P' such that (i) P is reachable from P' , (ii) $t_0 \in P'$ and $t_2 \notin P'$, and (iii) $P'_S = P_S = S$. Let $P'_1 = P'_T \setminus \{t_0\}$ and $P'_2 = P_S \setminus \{s_0, s_1\}$. Then, one can use the following transition:

$$(P'_1 \cup \{t_0, t_2\}) \cup (P'_2 \cup \{s_0\}) \xrightarrow{1} (P'_1 \cup \{t_0, s_0\}) \cup (P'_2 \cup \{s_1\}) = P',$$

where $(|Q_S|, |Q_T|) = (|P_S| - 1, |P_T| + 1)$. $\square$

We note that the proofs of Lemmas 4 and 5 do not depend on whether k is odd or even.

3.6 Inequivalence of Reachable f-States

We have so far shown that the number of reachable f-states in $D(M_1)$ is $2^{k+m} - (k+1) = 2^n - (k+1)$. Now we prove that those f-states are pair-wise inequivalent.

Lemma 6. *Any two reachable f-states of $D(M_1)$ are not equivalent.*

Proof. Let X and Y be two f-states such that $X \neq Y$. If $X_T \neq Y_T$, there must be an integer j such that $t_0 \in \delta(X_T, 0^j)$ and $t_0 \notin \delta(Y_T, 0^j)$. Thus X and Y are not equivalent. Next, suppose that $X_T = Y_T$ and $X_S \neq Y_S$. Then, there is an integer j such that $s_1 \in \delta(X_S, 0^j)$ ($= X'$) and $s_1 \notin \delta(Y_S, 0^j)$ ($= Y'$). We then read a 1, and $t_0 \in \delta(X', 1)$ while $t_0 \notin \delta(Y', 1)$. Therefore, $\delta(X', 1)$ and $\delta(Y', 1)$ have different T -portions and so are not equivalent, as shown previously. Hence, X and Y are not equivalent. $\square$

Table 2. Numbers of unreachable f-states

$\begin{array}{c c} \text{NFA} & P_T \end{array}$	0	1	2	3	4	$\cdots$	$k-1$	k	Total
M_1	1	0	k	0	0	$\cdots$	0	0	$k+1$
M_2	2	0	k	0	0	$\cdots$	0	0	$k+2$
M_3	1	0	k	k	0	$\cdots$	0	0	$2k+1$
M_4	2	0	k	k	0	$\cdots$	0	0	$2k+2$

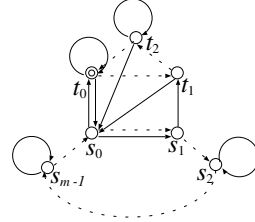


Fig. 2. M_5

3.7 Theorem 1 for $k = \alpha - 2$ and $\lceil \alpha/2 \rceil - 1$

We consider several modifications of M_1 to construct NFA's that realize other numbers of unreachable f-states. The modifications we consider are (i) to eliminate or add some transitions at M_1 , and (ii) to modify slightly some transitions of the type $t_i \xrightarrow{1} t_0$ to increase the number of unreachable states. Using the first type of modification, we obtain the following lemma.

Lemma 7. *Let M_2 be the NFA such that $s_0 \xrightarrow{1} t_0$ is added to M_1 and such that m is relatively prime with $k = \alpha - 2$. Then, the f-state S (i.e., $|P_T| = 0$ and $|P_S| = m$) is unreachable, while the reachability of the other f-states is the same as for M_1 .*

We omit the detailed proof but the intuition is as follows. Since $|P_T| = 0$ and $|P_S| = m$, we have to “remove” all T -states and “fill” all the S -states on reading the final 1. Previously, i.e., when there was no transition from s_0 to t_0 , we could do this by using $\{s_0, t_2\} \xrightarrow{1} \{s_0, s_1\}$. This is now impossible, since we have the transition $s_0 \xrightarrow{1} t_0$. $\square$

Using the second type of modification, we construct the NFA M_3 . M_3 has the transitions of the type $t_i \xrightarrow{1} t_0$ as follows. When k is odd, transitions $t_i \xrightarrow{1} t_0$ are defined for $i = 3$ and $i = 4, 6, 8, \dots, k-5$. When $k \bmod 4 = 0$, they are defined for $i = 3, 4, 6, 7, \dots, k-6, k-5$. When $k \bmod 4 = 2$, they are defined for $i = 3, 4, 6, 7, \dots, k-8, k-7, k-5$. Suppose that m is relatively prime with $k = \lceil \alpha/2 \rceil - 1$. Then with regard to unreachable f-states of M_3 , we obtain the following lemma.

Lemma 8. *In addition to the unreachable f-states for M_1 , M_3 has new unreachable f-states of the type $\{t_i, t_{i+3}, t_{i+4}\}$ ($0 \leq i \leq k-1$).*

The numbers of unreachable states for the M_i 's are summarized in Table 2.

Remark. Our assumption that k and m have no common divisor is necessary. For example, consider a simple case where $k = m$ or $|T| = |S|$. Then, $\{t_1, t_2, s_0\}$, which was formerly reachable, turns out to be unreachable.

4 NFA for Theorem 2

For the NFA M_5 given in Fig. 2, there are six unreachable f-states, i.e., ϕ , $S = \{s_0, s_1, \dots, s_{m-1}\}$, $\{t_0, t_1\}$, $\{t_1, t_2\}$, $\{t_2, t_0\}$, and $\{t_0, t_1, t_2\}$. Furthermore, all the reachable f-states are inequivalent; thus, $\Delta(M_5) = 2^{m+3} - 6 = 2^n - 6$. The proof for the reachability of f-states is similar to Theorem 1 except for the divisible case, i.e., $n \bmod 3 = 0$. In this case, we explicitly construct transitions for each f-state instead of using the coprimality condition and the 0-shifts.

5 Concluding Remarks

In this paper, we presented families of NFAs with n states, whose equivalent minimum DFAs have $2^n - \alpha$ states, subject to coprimality conditions on n and α . These NFAs are minimum since the equivalent DFAs have more than 2^{n-1} states. Finally, we conjecture that for all n there exists an n -state NFA M such that $\Delta(M) = 2^n - \alpha$ for any $0 \leq \alpha < 2^{n-1}$. To reach this range of α (without “holes” as in [5]) will need some new ideas.

Acknowledgments. We thank the anonymous referees for their helpful comments on our earlier version. We also gratefully acknowledge the help provided by the package *kbmag*, developed by Derek Holt (Department of Mathematics, University of Warwick), which contains a program for computing the minimum DFA equivalent to a given NFA. We used this program extensively during the design phase of our constructions; it enabled us to check potential NFAs for reasonable values of k and m , e.g., $k = 15$, $m = 10$.

References

1. M. Rabin and D. Scott, “Finite automata and their decision problems,” *IBM J. Res. Develop.* 3, pp. 114-125, 1959.
2. O. B. Lupanov, “Über den Vergleich zweier Typen endlicher Quellen,” *Probleme der Kybernetik*, Vol. 6, pp. 329-335, Akademie-Verlag, Berlin, 1966.
3. F. Moore, “On the bounds for state-set size in the proofs of equivalence between deterministic, nondeterministic, and two-way finite automata,” *IEEE Trans. Comput.* C-20, pp. 1211-1214, 1971.
4. J. E. Hopcroft and J. D. Ullman, *Introduction to automata theory, languages and computation*, Addison-Wesley, 1979.
5. K. Iwama, Y. Kambayashi, and K. Takaki, “Tight bounds on the number of states of DFA's that are equivalent to n -state NFA's,” *Theoretical Computer Science*, to appear. (<http://www.lab2.kuis.kyoto-u.ac.jp/~iwama/NfaDfa.ps>)

Preemptive Scheduling on Dedicated Processors: Applications of Fractional Graph Coloring

Klaus Jansen¹ and Lior Porkolab²

¹ Christian Albrechts University of Kiel, Germany, kj@informatik.uni-kiel.de

² Imperial College, London, UK, porkolab@doc.ic.ac.uk

Abstract. We study the problem of scheduling independent multiprocessor tasks, where for each task in addition to the processing time(s) there is a prespecified dedicated subset (or a family of alternative subsets) of processors which are required to process the task simultaneously. Focusing on problems where all required (alternative) subsets of processors have the same fixed cardinality, we present complexity results for computing preemptive schedules with minimum makespan closing the gap between computationally tractable and intractable instances. In particular, we show that for the dedicated version of the problem, optimal preemptive schedules of bi-processor tasks (i.e. tasks whose dedicated processor sets are all of cardinality two) can be computed in polynomial time. We give various extensions of this result including one to maximum lateness minimization with release times and due dates. All these results are based on a nice relation between preemptive scheduling and fractional coloring of graphs. In contrast to the positive results, we also prove that the problems of computing optimal preemptive schedules for three-processor tasks or for bi-processor tasks with (possible several) alternative modes are strongly NP-hard.

1 Introduction

In this paper we address multiprocessor scheduling problems, where a set of n tasks $\mathcal{T} = \{T_1, \dots, T_n\}$ has to be executed by m processors such that each processor can work on at most one task at a time, each task must be processed simultaneously by several processors, and processing of tasks can be interrupted any time at no cost. The objective discussed here is to minimize the makespan, i.e. the maximum completion time $C_{max} = \max\{C_j : T_j \in \mathcal{T}\}$. In the *dedicated* model denoted by $P|fix_j, pmtn|C_{max}$, each task requires the simultaneous use of a prespecified set of processors. In the *general* model $P|set_j, pmtn|C_{max}$, each task can have a number of *alternative modes*, where each processing mode is specified by a non-empty subset of processors and the execution time of the task on that particular processor set.

In one of the first papers on scheduling dedicated processors, Krawczyk and Kubale [11] showed that the non-preemptive version of the problem $P|fix_j|C_{max}$ is strongly NP-hard even if $|fix_j| = 2$, where the latter notation is used to indicate that each task T_j requires exactly two processors in parallel. Hoogeveen

et al. [9] also proved NP-hardness for the variant $P|fix_j, p_j = 1|C_{max} \leq 3$, where all processing times are of unit length. Furthermore, they showed that the problem $P3|fix_j|C_{max}$ with a constant number of processors is strongly NP-hard. Regarding the preemptive version of the problem, Kubale [12] proved that $P|fix_j, pmtn|C_{max}$ with $|fix_j| = 2$ is strongly NP-hard under the assumption that all events (starting times, interruptions, etc.) are at integer points in time. He also proved there that under the same assumptions $Pm|fix_j, pmtn|C_{max}$ with $|fix_j| = 2$ can be solved in time $O(\sum p_j)$.

In this paper, we study the computational complexity of the preemptive variant of the problem. But here - following the usual approach and assumptions in preemptive scheduling [4,5] - we do not assume (in contrast to [12]) that starting times and preemptions are at integer points in time. Interestingly, without this assumption, the preemptive scheduling problem leads to fractional coloring of graphs. Hence we obtain by using a general complexity result for linear programs in [8], along with a polynomial-time separation oracle, that $P|fix_j, pmtn|C_{max}$ with $|fix_j| = 2$ can be solved in polynomial time. This result can easily be extended to the case where $O(\log n)$ tasks are allowed to have dedicated processor sets of arbitrary cardinalities. Furthermore, we show the polynomial-time solvability of the more general problem $P|fix_j, pmtn|L_{max}$ with $|fix_j| = 2$, where in addition to the original input, there is also a due date d_j specified for each task T_j and the objective is to minimize the maximum lateness $L_{max} = \max\{C_j - d_j : T_j \in \mathcal{T}\}$. By combining further ideas on preemptive scheduling [2,3,10,13] with techniques for solving linear programs with exponentially many variables [8], one can also show that the previous polynomial-time complexity holds even for the problem $P|fix_j, pmtn, r_j|L_{max}$ with $|fix_j| = 2$, where both release times and due dates are specified for the tasks.

In contrast to these positive results for bi-processor tasks, we show that $P|fix_j, pmtn|C_{max}$ is strongly NP-hard when $|fix_j| = 3$ for all tasks. Finally, we prove that the more general problem $P|set_j, pmtn|C_{max}$ becomes strongly NP-hard even when $|set_j| = 2$, i.e. when for each task T_j , every alternative processing mode corresponds a processor set with cardinality of two.

2 Linear Programming Approach

Let $N = \{1, \dots, n\}$ and $M = \{1, \dots, m\}$. In problem $P|fix_j, pmtn|C_{max}$, each task $T_j \in \mathcal{T}$ requires the simultaneous use of a prespecified set $M(j) \subseteq M$ of processors. Two tasks T_i and T_j ($i \neq j$) can be processed at the same time, iff $M(i) \cap M(j) = \emptyset$. The processing conflicts can be modelled as an undirected graph $G = (V, E)$, where the vertex set $V = N$ corresponds to the set of tasks $\mathcal{T}$ and there is an edge $\{i, j\} \in E$, iff $M(i) \cap M(j) \neq \emptyset$ for the corresponding tasks T_i and T_j . Therefore independent sets in G correspond to subsets of tasks that can be processed in parallel. Thus problem $P|fix_j, pmtn|C_{max}$ consists of assigning a non-negative real value x_I to each independent set I of G (corresponding to processing times of the appropriate groups of processors working in parallel) such that each task is completely processed ("covered by independent sets") and

the total processing time $\sum_I x_I$ (which is also equal to the largest completion time) is minimized. This can also be formulated as the following linear program:

$$\begin{aligned} \text{Min } & \sum_{I \in \mathcal{F}} x_I \\ \text{s.t. } & \sum_{I: j \in I} x_I \geq p_j \quad \forall j \in N, \\ & x_I \geq 0 \quad \forall I \in \mathcal{F}, \end{aligned} \tag{1}$$

where p_j is the processing time of task T_j and $\mathcal{F}$ is the set of all independent sets in the conflict graph $G = (V, E)$. This problem is also called in the literature as *weighted fractional colouring*. Notice, that in general the number of independent sets, hence also the number of variables in (1), is exponentially large in n . Grötschel et al. [7] have shown that the problem is NP-hard for general graphs, but can be solved in polynomial time on perfect graphs. Specifically, they have proved the following result: For any graph class $\mathcal{G}$, if the problem of computing $\alpha(G)$, the size of the largest independent set in G , for graphs $G \in \mathcal{G}$ is NP-hard, then the problem of determining the weighted fractional chromatic number $\chi_f(G)$ is also NP-hard. This gives a negative result of the weighted fractional coloring problem even for planar cubic graphs. Furthermore, if the weighted independent set problem for graphs in $\mathcal{G}$ is polynomial-time solvable, then the weighted fractional coloring problem for $\mathcal{G}$ can also be solved in polynomial time.

In the following we will focus on the case when $|fix_j| = 2$ (there are only bi-processor tasks) and argue that problem $P|fix_j, pmtn|C_{max}$ becomes polynomial-time solvable under this assumption. First consider the dual of linear program (1):

$$\begin{aligned} \text{Max } & \sum_{j \in N} p_j y_j \\ \text{s.t. } & \sum_{j: j \in I} y_j \leq 1, \quad \forall I \in \mathcal{F}, \\ & y_j \geq 0, \quad \forall j \in N. \end{aligned} \tag{2}$$

This problem is also called the *weighted fractional clique problem*. The number of variables in (2) is polynomial in n , while the number of constraints is exponentially large. It is known [8], that if one can solve the *separation problem* for the dual in polynomial time, then one can also solve by the Ellipsoid Method the optimization problem of the primal in polynomial time. Therefore it suffices for us to consider the separation problem of (2): Given an n -vector y , decide whether y is a feasible solution of (2), and if not, find an inequality that separates y from the solution set of (2). This problem is equivalent to finding the maximum weighted independent set in G . For problems with only bi-processor tasks ($fix_j = 2$), one can consider a multi-graph $G' = (V', E')$, where vertices $i \in V'$ correspond to processors in M and edges $e \in E'$ represent the tasks. (There is an edge $\{i, j\}$ in E' for each task T whose required processor set consists of i and j .) Note, that a matching in G' corresponds to an independent set in G (hence also to a set of tasks that can be executed in parallel). Therefore the separation problem for instances with $|fix_j| = 2$ can be reduced to computing a maximum weighted matching in the multi-graph G' (with weights y_j from the given y vector). Since the latter problem is solvable in polynomial

time [7], there is also a polynomial-time algorithm for $P|fix_j, pmtn|C_{max}$ with $|fix_j| = 2$. Hence we have obtained the following result.

Theorem 1. *Problem $P|fix_j, pmtn|C_{max}$ with $|fix_j| = 2$ can be solved in polynomial time.*

It is natural to ask, whether this result remains true if there are a “few” exceptional tasks for which $|fix_j| \neq 2$. In fact, it does, hence Theorem 1 can be strengthened in the following way.

Theorem 2. *Problem $P|fix_j, pmtn|C_{max}$ can be solved in polynomial time when $|fix_j| = 2$ for all of the n tasks except for $O(\log n)$ (whose dedicated processor sets can be of arbitrary cardinality).*

Theorem 1 can also be extended to the more general variant of the problem, where for each task $T_j \in \mathcal{T}$ - in addition to the original input parameters - there is also a due date d_j given and the objective is to minimize the maximum lateness $L_{max} = \max\{C_j - d_j : j = 1, \dots, n\}$. Assume w.l.o.g. that the tasks are ordered according to their due dates, i.e. $d_1 \leq \dots \leq d_n$, and define the graph G as before. Let a configuration be a compatible subset of tasks that can be scheduled simultaneously and let $\mathcal{F}_\ell$ denote the set of all configurations that consists of only tasks from the subset $\{\ell, \ell + 1, \dots, n\}$, i.e. those which will be considered to be executable in the ℓ -th subinterval I_ℓ , where $I_1 = [d_0 \equiv 0, d_1 + L_{max}]$ and $I_\ell = [d_{\ell-1} + L_{max}, d_\ell + L_{max}]$ for $\ell = 2, \dots, n$. For $I \in \mathcal{F}_\ell$, let $x_I^{(\ell)}$ denote the length of configuration I in the ℓ -th subinterval of the schedule. Then the problem denoted by $P|fix_j, pmtn|L_{max}$ can be formulated as a linear program with variables $L_{max}, x_I^{(\ell)}$, for each $I \in \mathcal{F}_\ell, \ell \in N$, as follows:

$$\begin{aligned}
 & \text{Min } L_{max} \\
 & \text{s.t. } \sum_{I \in \mathcal{F}_1} x_I^{(1)} \leq d_1 - d_0 + L_{max}, \\
 & \quad \sum_{I \in \mathcal{F}_\ell} x_I^{(\ell)} \leq d_\ell - d_{\ell-1}, \quad \forall \ell \in N \setminus \{1\}, \\
 & \quad \sum_{\ell=1}^j \sum_{I \in \mathcal{F}_\ell: j \in I} x_I^{(\ell)} \geq p_j, \quad \forall j \in N, \\
 & \quad x_I^{(\ell)} \geq 0, \quad \forall I \in \mathcal{F}_\ell, \ell \in N.
 \end{aligned} \tag{3}$$

The corresponding dual linear program has the following form:

$$\begin{aligned}
 & \text{Max } \sum_{\ell=1}^n (d_{\ell-1} - d_\ell) y_\ell + \sum_{j=1}^n p_j z_j \\
 & \text{s.t. } y_1 = 1 \\
 & \quad \sum_{j \in I} z_j - y_\ell \leq 0, \quad \forall I \in \mathcal{F}_\ell, \ell \in N, \\
 & \quad y_\ell \geq 0, z_j \geq 0, \quad \forall j \in N, \ell \in N.
 \end{aligned} \tag{4}$$

This linear program has $2n$ variables and $O(n^{m+1})$ constraints. In the separation problem for (4), one has to decide whether a given vector $(y_1, \dots, y_n, z_1, \dots, z_n)$ is feasible for (4). Notice that this can be answered by considering the constraints in (4) for each $\ell \in \{1, \dots, n\}$ independently. For a given ℓ , one has to decide whether $\max_{I \in \mathcal{F}_\ell} \sum_{j \in I} z_j \leq y_\ell$, which can be done in polynomial time by solving the maximum weighted matching problem in the corresponding multi-graph G'_ℓ similarly as for C_{max} . Hence the following result holds.

Theorem 3. *Problem $P|fix_j, pmtn|L_{max}$ with $|fix_j| = 2$ can be solved in polynomial time.*

Suppose that there is also release date r_j specified for each task $T_j \in \mathcal{T}$. Labetoulle et al. [13] proposed an optimal polynomial time algorithm for problem $P|pmtn, r_j|L_{max}$ based on trial values for L_{max} . Combining ideas in [13] with techniques similar to those we have presented for the separation problem of (4), one can show the following result:

Theorem 4. *Problem $P|fix_j, pmtn, r_j|L_{max}$ with $|fix_j| = 2$ can be solved in polynomial time.*

Clearly, the two different extensions given in Theorems 2 and 4 of Theorem 1 can also be combined.

In closing this section we would like to further elaborate on the connection between our scheduling problem and fractional coloring (encapsulated by the above linear programming formulations), and use this to translate an interesting inapproximability result to $P|fix_j, pmtn|C_{max}$. Note that every undirected graph $G = (V, E)$ with $V = N$ can be obtained as a conflict graph corresponding to a scheduling problem with dedicated processors. The number of processors in this construction can be bounded by $|E|$, the number of edges in G . The idea is to define as processor set for a task the set of adjacent edges, i.e. if $E = \{e_1, \dots, e_m\}$ and $E(j)$ denotes the set of edges adjacent to vertex $j \in V$, then let $M = E$ and $M(j) = E(j) \subseteq E$ for task T_j . If all processing times are one, then we have the *fractional coloring problem*. Lund and Yannakakis [15] proved that there is a $\delta > 0$ such that there does not exist a polynomial time approximation algorithm for fractional coloring that achieves ratio n^δ , unless $P = NP$. This readily implies:

Proposition 1. *There exists $\delta > 0$ such that there is no polynomial-time approximation algorithm for $P|fix_j, pmtn|C_{max}$ (even if $p_j = 1$ for all $T_j \in \mathcal{T}$) with approximation ratio n^δ , unless $P = NP$.*

Lund and Yannakakis also noticed the following: If we consider the graph coloring problem as a special case of the set covering problem where the collection of sets is the collection of all independent sets of the graph (it is not listed explicitly), then the fractional chromatic number is the optimum value of the corresponding linear programming relaxation. The logarithmic bound given by the approximation algorithm of Lovasz [14] for the set covering problem bounds not only the ratio of the cost of the solution obtained to the cost of the optimal solution, but also to the cost of the best fractional cover. This is the best solution to the linear programming relaxation of the standard formulation of the set covering problem as an integer program. The coloring problem for an arbitrary undirected graph is harder to approximate. Feige and Kilian [6] proved that the chromatic number cannot be approximated within $\Omega(|V|^{1-\epsilon})$ for any $\epsilon > 0$, unless $ZPP = NP$. This also holds for fractional coloring, due to the logarithmic relationship of the chromatic and fractional chromatic numbers (i.e. $\chi_f(G) \leq \chi(G) \leq \chi_f(G)(1 + \ln \alpha(G))$). These imply the following result:

Proposition 2. *For any $\delta > 0$, $P|fix_j, pmtn|C_{max}$ has no polynomial-time approximation algorithm with approximation ratio $n^{1-\delta}$, unless $ZPP = NP$.*

3 Dedicated Model - Intractability Threshold at Three

The complexity of the problems discussed in the previous section changes when there are three prespecified processors for each task. In this case, the separation problem for (2) becomes NP-hard, since the three dimensional matching problem can be easily reduced to it. However even the following stronger result holds.

Theorem 5. *Problem $P|fix_j, pmtn|C_{max}$ with $|fix_j| = 3$ is NP-hard in the strong sense.*

Proof. We prove the statement in three steps. First, we show that the fractional coloring problem with $\chi_f(G) \leq 3$ is NP-hard. In the second step, we decrease the degree of each vertex in the graph to 4. Finally, we show that the generated graph in the second step is the conflict graph of a scheduling problem with three processors per task.

Step 1: We use a reduction from NAE-SAT: Given a set of clauses $C_1, \dots, C_m$ each with three literals using variables $x_1, \dots, x_n$, is there any truth assignment for the variables such that no clause has all literals true or all literals false. Each clause C_k consists of literals y_{k1}, y_{k2} and y_{k3} (that are negated or unnegated variables). We construct the graph $G = (V, E)$ as follows. Let

$$V = \{A\} \cup \{x_i, \bar{x}_i : 1 \leq i \leq n\} \cup \{C_{k\ell}, D_{k\ell} : 1 \leq k \leq m, 1 \leq \ell \leq 3\}.$$

The $3m$ vertices $C_{k\ell}$ correspond to the literals $y_{k\ell}$ in the clauses $C_1, \dots, C_m$. Furthermore, we use vertices $D_{k\ell}$ for the literals $\bar{y}_{k\ell}$ in the complement clauses $\bar{C}_k$ (i.e. each literal is replaced by its complement literal). The underlying idea is the following: If the original clauses $C_1, \dots, C_m$ have a truth assignment as described above (with at least one true and one false literal), then the new clauses $C_1, \dots, C_m, \bar{C}_1, \dots, \bar{C}_m$ also have such a truth assignment. Let the edge set E consist of triangles

$$\{A, x_i, \bar{x}_i\}, \quad i=1, \dots, n, \quad \text{and} \quad \{C_{k1}, C_{k2}, C_{k3}\}, \{D_{k1}, D_{k2}, D_{k3}\}, \quad k=1, \dots, m.$$

Furthermore, we have a set of connecting edges in E between variable vertices $x_i, \bar{x}_i$ and clause vertices $C_{k\ell}$. If $\bar{x}_i$ is the l -th literal in C_k , then we connect x_i with $C_{k\ell}$ and $\bar{x}_i$ with $D_{k\ell}$. If x_i is the l -th literal in C_k , then we connect $\bar{x}_i$ with $C_{k\ell}$ and x_i with $D_{k\ell}$. Note that the above reduction can be done in polynomial time.

Consider a fractional coloring with $\chi_f(G) \leq 3$. Since we have triangles in G , $\chi_f(G) = 3$. A fractional coloring corresponds to a collection of independent sets $U_1, \dots, U_a$ with positive values $z_1, \dots, z_a$ such that $\sum_{i=1}^a z_i = 3$ and that each vertex v is covered by at least 1: $\sum_{i:v \in U_i} z_i \geq 1$. Notice that since the LP (consisting of n constraints) also has a basic optimal solution, the number a of independent sets can be assumed to be bounded by n .

Consider now an independent set U in this collection that does not contain vertex A . Clearly, U must contain either x_i or $\bar{x}_i$ for $1 \leq i \leq 3$. If U does not contain x_i and $\bar{x}_i$ for an index i , $1 \leq i \leq n$, then the fractional chromatic number is larger than 3. Therefore, U contains a truth assignment. We set a variable x_i *true* (or *false*) if the vertex x_i (or $\bar{x}_i$) is in U . Now, we look at a clause with corresponding vertices C_{k1}, C_{k2} and C_{k3} . If each of these vertices is in conflict to a variable vertex in U , then again the fractional chromatic number is larger than 3. Therefore, exactly one of these three vertices is also in U . If all vertices C_{k1}, C_{k2} and C_{k3} are not in conflict to variable vertices in U , then all vertices D_{k1}, D_{k2} and D_{k3} are in conflict and, therefore, the fractional chromatic number is again > 3 . Therefore, U corresponds to a truth assignment with one true literal and one false literal in each clause. In fact, U contains two vertices $C_{k\ell}$ and $D_{k\ell'}$ with $\ell \neq \ell'$ for each $1 \leq k \leq m$.

For the other direction, consider a truth assignment with one literal $y_{i\ell_i}$ false and one literal $y_{i\ell'_i}$ true for each clause. Then, take one independent set U_1 with $\{x_i | x_i \text{ true}\} \cup \{\bar{x}_i | \bar{x}_i \text{ true}\}$. Furthermore, U_1 contains $C_{i\ell'_i}$ and $D_{i\ell_i}$ for $1 \leq i \leq m$. The second independent set U_2 consists of $\{x_i | x_i \text{ false}\} \cup \{\bar{x}_i | \bar{x}_i \text{ false}\}$ and $\{D_{i\ell'_i}, C_{i\ell_i} | 1 \leq i \leq m\}$. Finally, U_3 contains A and the remaining clause vertices.

Step 2: In this step, we reduce the degree of each vertex to four. We give a transformation from a graph G to a graph $\bar{G}$. For each vertex v in our graph G of degree $\delta(v)$ we construct a graph H_v (see also Figure 1).

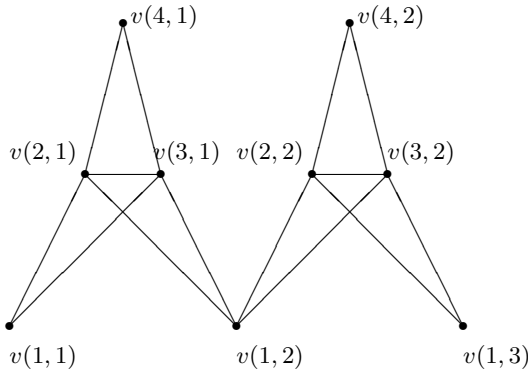


Fig. 1. Graph H_v for vertex $v \in V$.

The graph H_v with $a = \delta(v)$ has the vertex set

$$\{v(j, k) | 1 \leq j \leq 4, 1 \leq k \leq a\} \cup \{v(1, a+1)\}$$

and edge set

$$\begin{aligned} & \{(v(j, k), v(j', k)) : 1 \leq j \neq j' \leq 3, 1 \leq k \leq a\} \\ & \cup \{(v(j, k), v(j', k)) : 2 \leq j \neq j' \leq 4, 1 \leq k \leq a\} \\ & \cup \{(v(1, k), v(2, k-1)), (v(1, k), v(3, k-1)) : 2 \leq k \leq a+1\}. \end{aligned}$$

The graph H_v has $a + 2$ outlets, labelled by $v(1, 1)$, $v(1, a + 1)$ and $v(4, k)$ for $1 \leq k \leq a$. All these outlets have degree 2 and the other vertices have degree 4. In an ordinary 3-coloring, all outlets are colored with the same color. In a fractional coloring of value 3 of H_v , each independent set U contains either all outlets or no outlet. Furthermore, for each edge $e = \{v, w\}$ in the graph G , we connect the graphs H_v and H_w . To do this, we choose in both graphs H_v and H_w a free outlet (of degree 2) and connect these two outlets by an edge. The graph $\tilde{G}$ consists of the union of the graphs H_v for $v \in V$ plus the connecting edges. Then, we can prove the following statement: $\chi_f(G) \leq 3$, if and only if $\chi_f(\tilde{G}) \leq 3$.

Step 3: The last step is to show that the graph constructed in Step 2 is the conflict graph of an instance of $P|fix_j, pmtn|C_{max}$ with $|fix_j| = 3$. The main idea is to label the edges of each graph H_v by $1, 2, \dots$ and then to choose the processor sets for the vertices in H_v according to the adjacent edges. (The details for this step and hence the rest of the proof will be presented in the full version of the paper.) $\square$

4 General Model - Intractability Threshold at Two

In this section, we consider preemptive scheduling of tasks with different alternative processor sets. Suppose that for each task T_j , there are given subsets of processors $M_1(j), \dots, M_{s_j}(j) \subseteq M$ each with cardinality of two such that any of them can be used to process task T_j . At any time during the schedule when T_j is active, one of these sets have to be selected for its processing. There are two variants of the problem depending on whether migration is allowed or not:

- with no migration: for each task, one of its feasible processor sets has to be selected and used during the entire schedule;
- with migration: different feasible processor sets can be used for each task during the schedule.

The following result holds for both of these variants.

Theorem 6. *Problem $P|set_j, pmtn|C_{max}$ with $|set_j| = 2$ is NP-hard in the strong sense.*

Proof. We prove this by a reduction from the satisfiability problem SAT: Given a set of clauses $C_1, \dots, C_m$, is there a truth assignment for the variables $x_1, \dots, x_n$ such that each clause has at least one literal with value true. We will use a restricted (still NP-complete) variant of this problem, where each clause C_k has two or three literals $y_{k\ell}$, each variable x_i , occurs in exactly four clauses and each variable occurs twice negated and twice unnegated.

First, we show that this restricted problem is NP-complete. In the first step, we reduce all variables that occur only once. Then, if a variable x occurs α times with $\alpha \geq 2$, we introduce a cycle $(z_1 \vee \bar{z}_2) \dots (z_\alpha \vee \bar{z}_1)$ with new variables z_i , $1 \leq i \leq \alpha$ and replace the ℓ -th occurrence of x by z_ℓ . After this step, each variable occurs three times. For each missing literals $y \in \{x_i, \bar{x}_i\}$ we add four

clauses $(y \vee a \vee \bar{b})$, $(\bar{a} \vee b)$, $(\bar{a} \vee b)$ and $(a \vee \bar{b})$. After this step, all variables occur four times and twice negated and twice unnegated.

In our constructed instance, we will have $8n$ processors (eight processors $8(i-1)+1, \dots, 8i$ for each variable x_i) and $n+m$ tasks (a task for each variable and each clause). The task T_i ($1 \leq i \leq n$) can be executed on the processor sets $\{8(i-1)+1, 8(i-1)+2\}$ or $\{8(i-1)+3, 8(i-1)+4\}$. The first processor set is called mode $\bar{x}_i$ and the second is called mode x_i . A variable task T_i is either executed in mode x_i or mode $\bar{x}_i$.

The clause tasks $T_{n+1}, \dots, T_{n+m}$ can be executed in two or three modes (depending on the number of literals in the clause). Let us consider the case $y_{k\ell} \in \{x_i, \bar{x}_i\}$ for $1 \leq \ell \leq 3$. The construction for a clause with two literals works as well. The literal $y_{k\ell}$ corresponds to a mode in which task T_{n+k} can be processed. If $y_{k\ell} = x_i$ and this is the ℓ -th occurrence of x_i in a clause ($\ell \in \{1, 2\}$), then T_{n+k} can be executed on the processor set $\{8(i-1)+\ell, 8(i-1)+\ell+4\}$. If $y_{k\ell} = \bar{x}_i$ and this is the ℓ -th occurrence of $\bar{x}_i$ in a clause ($\ell \in \{1, 2\}$), then T_{n+k} can be executed on the processor set $\{8(i-1)+2+\ell, 8(i-1)+6+\ell\}$. The construction for x_{i+1} and the corresponding clauses are shown in Figure 2.

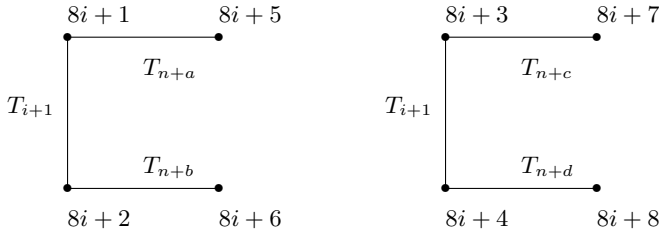


Fig. 2. Construction for x_i , where $x_{i+1} \in C_a, C_b$ ($a < b$) and $\bar{x}_{i+1} \in C_c, C_d$ ($c < d$).

The processing times of all tasks are 1 and $C_{max} = 1$. Then, we can prove that there is a truth assignment that satisfies all clauses if and only if there is a preemptive schedule for the constructed instance with makespan 1.

Let $\phi : \{x_i | 1 \leq i \leq n\} \rightarrow \{true, false\}$ be a truth assignment that satisfies all clauses. If x_i is true, then we choose as mode for T_i the processor set $\{8(i-1)+3, 8(i-1)+4\}$. If x_i is false, then we choose $\{8(i-1)+1, 8(i-1)+2\}$. Since ϕ satisfies all clauses, there is at least one literal $y_{k\ell_k}$ for each clause C_k with value true. For T_{n+k} we choose in this case the mode that corresponds to this literal. The processor set (corresponding to this mode) is not used by other tasks. Therefore, we can execute all tasks in parallel without preemptions and get a schedule with makespan 1.

For the other direction, consider a schedule with makespan 1. Then take a set of tasks U that is executed in parallel in this schedule. For each variable x_i , task T_i is in the U (either executed in mode x_i or $\bar{x}_i$), since otherwise the total length of the schedule would be larger than 1. If T_i is executed in mode x_i , let

x_i be true, and otherwise false. If a clause task T_{n+k} were missing from U , then again the length of the schedule would be larger than 1. Therefore all clause tasks $T_{n+1}, \dots, T_{n+m}$ are in U . Since the chosen modes of the variable tasks block modes of clauses with corresponding truth value false, for each clause C_k the chosen mode (or processor set) corresponds to a truth value true. $\square$

References

1. L. Bianco, J. Blazewicz, P. Dell'Olmo and M. Drozdowski, Scheduling preemptive multiprocessor tasks on dedicated processors, *Performance Evaluation* 20 (1994), 361-371.
2. L. Bianco, J. Blazewicz, P. Dell'Olmo and M. Drozdowski, Preemptive multiprocessor task scheduling with release and time windows, *Annals of Operations Research* 70 (1997), 43-55.
3. J. Blazewicz, M. Drozdowski, D. de Werra and J. Weglarz, Deadline scheduling of multiprocessor tasks, *Discrete Applied Mathematics* 65 (1996), 81-95.
4. J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt and J. Weglarz, Scheduling Computer and Manufacturing Processes, Springer Verlag, Berlin, 1996.
5. M. Drozdowski, Scheduling multiprocessor tasks - an overview, *European Journal on Operations Research*, 94 (1996), 215-230.
6. U. Feige and J. Kilian, Zero knowledge and the chromatic number, *Journal of Computer and System Sciences*, 57 (1998), 187-199.
7. M. Grötschel, L. Lovasz and A. Schrijver, The Ellipsoid Method and its consequences in combinatorial optimization, *Combinatorica*, 1 (1981) 169-197.
8. M. Grötschel, L. Lovasz and A. Schrijver, Geometric Algorithms and Combinatorial Optimization, Springer Verlag, Berlin, 1988.
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. A. Krämer, Scheduling multiprocessor tasks on dedicated processors, Ph.D.thesis, Fachbereich Mathematik - Informatik, Universität Osnabrück, Germany, 1995.
11. H. Krawczyk and M. Kubale, An approximation algorithm for diagnostic test scheduling in multicomputer systems, *IEEE Transactions on Computers* 34 (1985), 869-872.
12. M. Kubale, Preemptive versus nonpreemptive scheduling of biprocessor tasks on dedicated processors, *European Journal of Operational Research*, 94 (1996), 242-251.
13. J. Labetoulle, E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan, Preemptive scheduling of uniform machines subject to release dates, in: W.R. Pulleyblank (ed.), *Progress in Combinatorial Optimization*, Academic Press, New York, 1984, 245-261.
14. L. Lovasz, On the ratio of optimal integral and fractional covers, *Discrete Mathematics* 13 (1975), 383-390.
15. C. Lund and M. Yannakakis, On the hardness of approximating minimization problems, *Journal of the ACM* 41 (1994), 960-981.

Matching Modulo Associativity and Idempotency Is NP-Complete*

Ondřej Klíma¹ and Jiří Srba²

¹ Faculty of Science MU, Dept. of Mathematics, Janáčkovo nám. 2a, 662 95 Brno, Czech Republic, klima@math.muni.cz

² BRICS***, Department of Computer Science, University of Aarhus, Ny Munkegade bld. 540, DK-8000 Aarhus C, Denmark, srba@brics.dk

Abstract. We show that AI-matching (AI denotes the theory of an associative and idempotent function symbol), which is solving matching word equations in free idempotent semigroups, is NP-complete.

Note: full version of the paper appears as [8].

1 Introduction

Solving equations appears as an interesting topic in several fields of computer science. Many areas such as logic programming and automated theorem proving exploit solving equations, and syntactic (Robinson) unification is a typical example of it. An important role is played also by semantic unification, which allows to use several function symbols with additional algebraic properties (e.g. associativity, commutativity and idempotency). Makanin (see [15]) shows that the question whether an equation in a free monoid has a solution is decidable. It can be generalized in the way that existential first-order theory of equations in a free monoid with additional regular constraints on the variables is decidable [18]. For an overview of unification theory consult e.g. [4].

AI-matching is one example of semantic unification where the considered equational theory is of one associative and idempotent function symbol. In this paper we focus on a subclass of word equations which we call pattern equations. Pattern equations are word equations where we have on the left-hand side just variables and on the right-hand side only constants. In the usual interpretation, AI-matching is a AI-unification of systems of equations where all right-hand sides are variable-free. However, we can eliminate constants on the left-hand sides by adding new equations and so pattern equations are as general as AI-matching.

Many practical problems such as speech recognition/synthesis lead to this kind of equations. This work has been inspired by the papers [11] and [12] where the basic approach – syllable-based speech synthesis – is in assigning prosody

* The paper is supported by the Grant Agency of the Czech Republic, grant No. 201/97/0456 and by grant FRVŠ 409/1999.

*** Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

attributes to a given text and segmentation into syllable segments. We examine the solvability of word equations in the variety of all idempotent semigroups, which we call stuttering equations. The decidability of the satisfiability problem (even in the general case) is a consequence of the local finiteness of free idempotent semigroups and an exponential upper bound on the length of a minimal solution can be given ([6]). A polynomial time decision procedure for the word problem in a free idempotent semigroup can be also easily established. Recently it has been proved in [3] that AI-unification remains decidable even if additional uninterpreted function symbols in the equations are allowed.

Unification problems for the AI-theory have been investigated e.g. in [1,2,19], however, the complexity questions were not answered. In this paper we prove that there is a polynomial bound on the length of a minimal solution in the case of stuttering pattern equations and thus we show that the satisfiability problem is in NP. The proof exploits the confluent and terminating word rewriting system for idempotent semigroups by Siekmann and Szabo (see [20]). This means that the identity $p = q$ holds in a free idempotent semigroup if and only if the words p and q have the same normal form w.r.t. the rewriting system $\{xx \rightarrow x \mid C(x) \neq \emptyset\} \cup \{uvw \rightarrow uw \mid \emptyset \neq C(v) \subseteq C(u) = C(w)\}$, where $C(y)$ denotes the set of letters of y .

Showing a reduction from 3-SAT to our problem, we prove its NP-completeness. This is a more general result than Theorem 7 in the paper by Kapur and Narendran [10], where they prove NP-hardness for AI-matching, where additional uninterpreted function symbols are allowed. In our proof we use only one associative and idempotent function symbol.

Many proofs in this paper are not included due to space limitation and the full version can be obtained as [8].

2 Basic Definitions

An idempotent semigroup (also called a band) is a semigroup where the identity $x^2 = x$ is satisfied. Let $\mathcal{C}$ be a finite set. We define a binary relation $\rightarrow \subseteq \mathcal{C}^* \times \mathcal{C}^*$ such that $uvvw \rightarrow uvw$ for $u, v, w \in \mathcal{C}^*$ and let $\sim$ be its symmetric, reflexive and transitive closure, i.e. $\sim := (\rightarrow \cup \rightarrow^{-1})^*$. Then the identity $p = q$ holds in a free band over $\mathcal{C}$ if and only if $p \sim q$ (completeness of the equational logic).

Let $\mathcal{C}$ be a finite set of *constants* and $\mathcal{V}$ be a finite set of *variables* such that $\mathcal{C} \cap \mathcal{V} = \emptyset$. A *word equation* $L = R$ is a pair $(L, R) \in (\mathcal{C} \cup \mathcal{V})^* \times (\mathcal{C} \cup \mathcal{V})^*$. A *system of word equations* is a finite set of equations of the form $\{L_1 = R_1, \dots, L_n = R_n\}$ for $n > 0$. A *solution (in a free idempotent semigroup)* of such a system is a homomorphism $\alpha : (\mathcal{C} \cup \mathcal{V})^* \rightarrow \mathcal{C}^*$ which behaves as an identity on the letters from $\mathcal{C}$ and equates all the equations of the system, i.e. $\alpha(L_i) \sim \alpha(R_i)$ for all $1 \leq i \leq n$. Such a homomorphism is fully established by a mapping $\alpha : \mathcal{V} \rightarrow \mathcal{C}^*$. A solution is called *non-singular*, if $\alpha(x) \neq \epsilon$ for all $x \in \mathcal{V}$, where ϵ denotes the empty word. Otherwise we will call it *singular*. We say that a system of word equations (in a free idempotent semigroup) is *satisfiable* whenever it has a solution. For the introduction into word equations and combinatorics on words

you can see [13], [14] and [17]. We refer to word equations in a free idempotent semigroup as *stuttering equations*.

In what follows we will use a uniform notation. The set $\mathcal{C} = \{a, b, c, \dots\}$ denotes the alphabet of constants and $\mathcal{V} = \{x, y, z, \dots\}$ stands for variables (unknowns) with the assumption that $\mathcal{C} \cap \mathcal{V} = \emptyset$. We will use the same symbol α for the mapping $\alpha : \mathcal{V} \rightarrow \mathcal{C}^*$ and its unique extension to a homomorphism $\alpha : (\mathcal{C} \cup \mathcal{V})^* \rightarrow \mathcal{C}^*$. The empty word will be denoted by ϵ and the length of a word w by $|w|$.

We exploit the fact that the word problem in a free band is decidable (see [7] and its generalization [9]), which is a consequence of the next lemma. Let $w \in \mathcal{C}^+$. We define $\mathcal{C}(w)$ – the set of all letters that occur in w ; $0(w)$ – the longest prefix of w in $\text{card}(\mathcal{C}(w)) - 1$ letters; $1(w)$ – the longest suffix of w in $\text{card}(\mathcal{C}(w)) - 1$ letters. Let also $\bar{0}(w)$ resp. $\bar{1}(w)$ be the letter that immediately succeeds $0(w)$ resp. precedes $1(w)$.

Lemma 1 ([7]). *Let $p, q \in \mathcal{C}^+$. Then $p \sim q$ if and only if $\mathcal{C}(p) = \mathcal{C}(q)$, $0(p) \sim 0(q)$ and $1(p) \sim 1(q)$.*

Lemma 2. *Let $\{L_1 = R_1, \dots, L_n = R_n\}$ be a stuttering equation system with a solution α . Then also any β that satisfies $\alpha(x) \sim \beta(x)$ for all $x \in \mathcal{V}$ (we simply write $\alpha \sim \beta$) is a solution.*

This gives an idea that we should look just for the solutions where $\alpha(x)$ is the shortest word in the $\sim$ class for each variable x . We introduce a size of a solution α as $\text{size}(\alpha) := \max_{x \in \mathcal{V}} |\alpha(x)|$ and say that α is *minimal* iff for any solution β of the system we have $\text{size}(\alpha) \leq \text{size}(\beta)$. Given a stuttering equation system it is decidable whether the system is satisfiable because of the local finiteness of free idempotent semigroups. The following lemma just gives a precise exponential upper bound on the size of a minimal solution.

Lemma 3 ([6]). *Let $k = \text{card}(\mathcal{C}) \geq 2$ and let $\{L_1 = R_1, \dots, L_n = R_n\}$ be a stuttering equation system. If the system is satisfiable then there exists a solution α such that $\text{size}(\alpha) \leq 2^k + 2^{k-2} - 2$.*

In general it can be shown that there are stuttering equation systems such that all their solutions are at least exponentially large w.r.t. the cardinality of the set $\mathcal{C}$. Consider the following sequence of equations: $x_1 = a_1$ and $x_{i+1} = x_i a_{i+1} x_i$ for a sequence of pairwise different constants $a_1, a_2, \dots$. For any solution α of the system we have that $|\alpha(x_i)| \geq 2^i - 1$. In this paper we focus on a special kind of word equations which we call *pattern equations*.

Definition 1. *A pattern equation system is a set $\{X_1 = A_1, \dots, X_n = A_n\}$ where $X_i \in \mathcal{V}^*$ and $A_i \in \mathcal{C}^*$ for all $1 \leq i \leq n$. A solution of a pattern equation system is defined as in the general case.*

Remark 1. In the usual interpretation AI-matching allows constants to appear also on the left-hand sides, i.e. the equations are of the type $X = A$ where $X \in (\mathcal{V} \cup \mathcal{C})^*$ and $A \in \mathcal{C}^*$. However, we can w.l.o.g. consider only pattern equations, since an equation of the type $X_1 a X_2 = A$ where $a \in \mathcal{C}$ can be transformed into $X_1 x X_2 = A$ and $x = a$, where x is a new variable.

Definition 2. *Given a pattern equation system $\{X_1 = A_1, \dots, X_n = A_n\}$ as an instance of the PATTERN-EQUATION problem, the task is to decide whether this system has a solution. If we require the solution to be non-singular we call the problem NON-SINGULAR-PATTERN-EQUATION.*

The PATTERN-EQUATION problem for a single *stuttering* pattern equation $X = A$ is trivial since it is always solvable: $\alpha(x) = A$ for all $x \in \mathcal{V}$. On the other hand a system is not always solvable: e.g. $\{x = a, x = b\}$ has no solution. Our goal is to show that a minimal solution is of a polynomial length.

3 Rewriting System for Idempotent Semigroups

In this section we summarize several properties of the rewriting system by Siekmann and Szabo in [20] and give some technical lemmas. First of all we have to give some definitions and results concerning rewriting systems (see e.g. [5]).

A *rewriting system* R over $\mathcal{C}$ is a subset of $\mathcal{C}^* \times \mathcal{C}^*$. The elements of R will be called *rules*. Having such a system R we can define a *rewrite relation* $\rightarrow \subseteq \mathcal{C}^* \times \mathcal{C}^*$.

$$\forall p, q \in \mathcal{C}^* : p \rightarrow q \text{ iff } \exists (u, v) \in R, s, t \in \mathcal{C}^* : p = sut, q = svt$$

The elements (u, v) of R will be often written as $u \rightarrow v$. For a word $q \in \mathcal{C}^*$ we write $q \not\rightarrow$ iff there is no q' such that $q \rightarrow q'$ and we say that q is in a *normal form*. We define the set of normal forms of $p \in \mathcal{C}^*$ as $\langle p \rangle = \{q \mid p \rightarrow^* q \not\rightarrow\}$. We say that R (resp. the relation $\rightarrow$) is *terminating* iff there is no infinite sequence $p_1, p_2, p_3, \dots \in \mathcal{C}^*$ such that $p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow \dots$. The system R (resp. the relation $\rightarrow$) is *confluent* iff

$$\forall p, p_1, p_2 \in \mathcal{C}^* \exists q \in \mathcal{C}^* : \text{ if } (p \rightarrow^* p_1 \text{ and } p \rightarrow^* p_2) \text{ then } (p_1 \rightarrow^* q \text{ and } p_2 \rightarrow^* q).$$

The system R (resp. the relation $\rightarrow$) is *locally confluent* iff

$$\forall p, p_1, p_2 \in \mathcal{C}^* \exists q \in \mathcal{C}^* : \text{ if } (p \rightarrow p_1 \text{ and } p \rightarrow p_2) \text{ then } (p_1 \rightarrow^* q \text{ and } p_2 \rightarrow^* q).$$

Lemma 4 ([5]). *Let R be a terminating rewriting system. Then R is confluent if and only if R is locally confluent.*

It is easy to see that if R is a confluent and terminating rewriting system, then a word $p \in \mathcal{C}^*$ has exactly one normal form, i.e. $\langle p \rangle = \{q\}$ for some q , and in such a case we simply write $\langle p \rangle = q$. In this paper we will exploit the rewriting system by Siekmann and Szabo in [20].

Lemma 5 ([20]). *The rewriting system $\{xx \rightarrow x \mid x \in \mathcal{C}^*, \mathcal{C}(x) \neq \emptyset\} \cup \{uvw \rightarrow uw \mid u, v, w \in \mathcal{C}^*, \emptyset \neq \mathcal{C}(v) \subseteq \mathcal{C}(u) = \mathcal{C}(w)\}$ is confluent and terminating. Moreover for $p, q \in \mathcal{C}^*$ we have $p \sim q$ if and only if p and q have the same normal form w.r.t. the system.*

We will refer the rewriting system $\{xx \rightarrow x \mid \mathcal{C}(x) \neq \emptyset\} \cup \{uvw \rightarrow uw \mid \emptyset \neq \mathcal{C}(v) \subseteq \mathcal{C}(u) = \mathcal{C}(w)\}$ as *RS*. Since *RS* contains two different types of rewriting rules we denote *RS*₁ the rewriting system $\{xx \rightarrow x \mid \mathcal{C}(x) \neq \emptyset\}$ and *RS*₂ the

rewriting system $\{uvw \rightarrow uw \mid \emptyset \neq C(v) \subseteq C(u) = C(w)\}$. The corresponding rewrite relations are denoted $\rightarrow$, $\rightarrow_1$ resp. $\rightarrow_2$ and for a word $p \in \mathcal{C}^*$ the set of its normal forms is denoted by $\langle p \rangle$, $\langle p \rangle_1$ resp. $\langle p \rangle_2$.

If we want to investigate the complexity issues for stuttering equations, the first question we have to answer is the complexity of checking whether some identity holds in a free band. It can be easily shown that the word problem (i.e. the problem whether $p \sim q$) can be decided in polynomial time by using the rewriting system RS .

We know that RS is confluent and terminating. Our goal in this section is to show that RS_2 is also a confluent and terminating rewriting system and that $\langle p \rangle = \langle \langle p \rangle_2 \rangle_1$.

We define a rewrite relation $\rightarrow_{2l} \subset \rightarrow_2$ such that $suvt \rightarrow_{2l} suwt$ if and only if $|v| = 1$ and $C(v) \subseteq C(u) = C(w)$. It is easy to see that $\rightarrow_2 \subseteq \rightarrow_{2l}^*$ and hence $\rightarrow_{2l}^* = \rightarrow_2^*$. The last relation we will use is $\rightarrow_{2m} \subset \rightarrow_2$, consisting of all rules that leave out the maximal number of letters in the following sense. Let $H(w)$ resp. $T(w)$ mean the first resp. the last letter of the word w . We write $suvt \rightarrow_{2m} suwt$ if and only if $\emptyset \neq C(v) \subseteq C(u) = C(w)$ and the following conditions hold:

- (i) $C(s_0u) \neq C(wt_0)$, for any suffix s_0 of s and any prefix t_0 of t (including empty s_0 or t_0 , but not both)
- (ii) $u = 0(u)T(u)$
- (iii) $w = H(w)1(w)$.

Note that if $suvt \rightarrow_{2m} suwt$ then the last letter of s and the first letter of t (if they exist) are new and different letters¹. Also note that $T(u)$ is the only occurrence of this letter in u and we can write it as $u = 0(u)\bar{0}(u)$. Similarly $w = \bar{1}(w)1(w)$. We remind that whenever $\rightarrow_2$ rewriting applies then so does $\rightarrow_{2m}$ and $\rightarrow_{2l}$. Moreover a word is in normal form w.r.t. $\rightarrow_2$ iff it is in normal form w.r.t. $\rightarrow_{2m}$ and iff it is in normal form w.r.t. $\rightarrow_{2l}$. In what follows, we will use these trivial observations without any explicit reference.

We show that $\langle p \rangle_{2m} = \langle p \rangle_2$. The inclusion $\langle p \rangle_{2m} \subseteq \langle p \rangle_2$ is obvious and the rest is the content of the following lemmas. We use the notation $suvt \rightarrow_2 suwt$ in the sense that $suvt \rightarrow_2 suwt$ where $\emptyset \neq C(v) \subseteq C(u) = C(w)$ (and the same for $\rightarrow_{2l}$, $\rightarrow_{2m}$). In the following, whenever we say that u is a subword of sut , we always refer to the concrete (and obvious) occurrence of the subword u in sut .

Lemma 6. *The relation $\rightarrow_{2m}$ is confluent and terminating.*

Remark 2. Two arbitrary applications of $\rightarrow_{2m}$, say $p = s_1u_1v_1w_1t_1 \rightarrow_{2m} s_1u_1w_1t_1$ and $p = s_2u_2v_2w_2t_2 \rightarrow_{2m} s_2u_2w_2t_2$, commute (i.e. they are independent of the order of their applications) and they can be nested exactly in one of the following ways (up to symmetry):

1. $w_1 = w'_1q$, $u_2 = qu'_2$ and $p = s_1u_1v_1w'_1qu'_2v_2w_2t_2$
2. $u_1v_1w_1$ is a subword of u_2

¹ Observe that it doesn't hold in general that if $p \rightarrow_{2m} q$ then $spt \rightarrow_{2m} sqt$ for $s, t \in \mathcal{C}^*$. This means that $\rightarrow_{2m}$ is not a rewriting relation in the previously introduced sense.

Lemma 7. RS_2 is a confluent and terminating rewriting system and $\langle p \rangle_{2m} = \langle p \rangle_2$ for any $p \in \mathcal{C}^*$.

Lemma 8. For any $p, q \in \mathcal{C}^*$ such that $p = \langle p \rangle_2$ and $p \rightarrow_1 q$ it holds that $\langle q \rangle_2 = q$. In particular for a word $p \in \mathcal{C}^*$ we have $\langle \langle p \rangle_2 \rangle_1 = \langle p \rangle$.

4 Upper Bound for the Size of the Solution

This section aims to prove that the PATTERN-EQUATION problem is in NP by giving a polynomial upper bound on the size of a minimal solution. In the following we assume implicitly that $A, B \in \mathcal{C}^*$. Realise that each reduction of RS just leaves out some subword, the case $uvw \rightarrow uv$ is clear, and in the case $xx \rightarrow x$ we leave out the right occurrence of x in the square. If we have a word uAv , we can speak about the *residual* of A in the sense that the residual consists of all letter occurrences of A that were not left out during the sequence of reductions. Moreover if we use two different sequences of reductions by $\rightarrow_{2m}$, which give normal form w.r.t. $\rightarrow_2$, then the residuals are the same after the both reduction sequences, since any two applications of $\rightarrow_{2m}$ commute by Remark 2.

Lemma 9. Let A and B be in normal form and $AB \rightarrow_{2m} AB' \not\rightarrow_{2m}$ where B' is the residual of B . Then the word B' contains at most one square x^2 . If B' contains such a square, then x^2 arises from xvx where v is the word left out by the reduction rule $\underline{uvw} \rightarrow_{2m} uw$, and x is both a suffix of u and a prefix of w . Moreover in the case when B' contains a square we have $B' \rightarrow_1 \langle B' \rangle$.

Remark 3. The previous lemma has the following analogue. If A_1, B, A_2 are in normal form and $A_1BA_2 \rightarrow_{2m} A_1B'A_2 \not\rightarrow_{2m}$ and if the residual B' of B contains a square x^2 , then $B' \rightarrow_1 \langle B' \rangle_1$ and x has the same properties as in Lemma 9.

Proposition 1. Let A and B be in normal form such that $\langle AB \rangle_2 = AB'$ where B' is the residual of B , then $|B'| \leq |\langle B' \rangle|^2$.

Proof. By Lemma 7 we have $\langle AB \rangle_2 = \langle AB \rangle_{2m}$ and we can use the maximal reductions. W.l.o.g. assume that the reductions $\rightarrow_{2m}$ did not leave out some prefix B_1 of the word B , otherwise we can start with the words A and B_2 where $B = B_1B_2$. Remark 2 shows how two applications of $\rightarrow_{2m}$ can be nested. Since A and B are in normal form, we can see that any reduction $\rightarrow_{2m}$ uses some letters from both A and B . Since A is left untouched, we can write $A = s_{n+1}s_n \dots s_1$, $B = u_1v_1w_1 \dots u_nv_nw_nu_{n+1}$ where $s_i, u_i, v_i, w_i \in \mathcal{C}^*$ for all possible i and we have n reductions of the form

$$s_{n+1} \dots \underline{s_i \dots s_1 u_1 w_1 \dots u_i v_i w_i} \dots u_{n+1} \rightarrow_{2m} s_{n+1} \dots s_1 u_1 w_1 \dots u_i w_i \dots u_{n+1}$$

where $C(v_i) \subseteq C(s_i \dots s_1 u_1 w_1 \dots u_i) = C(w_i)$ and $B' = u_1 w_1 \dots u_n w_n u_{n+1}$.

Since each step of the maximal reduction needs a new letter (the letter that immediately succeeds w_i), we get an upper bound for n (the number of

steps in $\rightarrow_{2m}^*$, $n+1 \leq \text{card}(\mathcal{C}(B))$. Let us denote $B'' = \langle B' \rangle_1 = \langle B' \rangle$ and $w_0 = \epsilon$. By induction (where $i = 1, \dots, n$) and by Lemma 9 applied on A and $\langle w_0 u_1 \dots w_{i-1} u_i \rangle v_i w_i u_{i+1}$ we can see that $|B''| \geq \max_{i=1}^{n+1} \{|w_{i-1} u_i|\}$ since after every application $xx \rightarrow x$ we can find each $w_{i-1} u_i$ as a subword in the residual of B . Hence we get $|B''| \geq \max_{i=1}^{n+1} \{|w_{i-1} u_i|\} \geq \frac{1}{n+1} \sum_{i=1}^{n+1} |w_{i-1} u_i| = \frac{|B'|}{n+1}$ and from the fact $n+1 \leq \text{card}(\mathcal{C}(B)) = \text{card}(\mathcal{C}(B''))$ we can deduce that $|\langle B' \rangle|^2 = |B''|^2 \geq \text{card}(\mathcal{C}(B'')) \cdot |B''| \geq (n+1) \frac{|B'|}{n+1} = |B'|$. $\square$

The previous proposition can be generalized in the following way.

Proposition 2. *Let A_1, B and A_2 be in normal form and $\langle A_1 B A_2 \rangle_2 = A'_1 B' A'_2$ where A'_1, B', A'_2 are the residuals of A_1, B, A_2 . Then $|B'| \leq 2 \cdot |\langle B' \rangle|^2$.*

Lemma 10. *Let $sxxt$ be a word such that $\langle sxxt \rangle_2 = sxxt$ and $sxxt$ contains another square y^2 ($|y| \leq |x|$) such that one of these occurrences of y lies inside the x^2 . Then one of the following conditions holds:*

1. y is a suffix of s and a prefix of x
2. y is a prefix of t and a suffix of x
3. y^2 is a subword of x

Remark 4. The previous lemma shows that for two applications of the rules $xx \rightarrow_1 x$ and $yy \rightarrow_1 y$ on a word p in normal form w.r.t. $\rightarrow_2$, one of the following conditions holds (up to symmetry):

1. xx and yy do not overlap
2. yy is a subword of x
3. $x = x'z, y = zy'$ and $xx'zy'y$ is a subword of p

Lemma 11. *If $\langle sxxt \rangle_2 = sxxt$ and sxt contains a square y^2 which is not in $sxxt$ then $y = s_1 x t_1$ where $|s_1|, |t_1| \geq 1$.*

Proposition 3. *Let A and B be in normal form, $\text{card}(\mathcal{C}(AB)) \geq 2$ and $\langle AB \rangle_2 = AB$. Then $|AB| \leq |\langle AB \rangle|^2$.*

Proposition 4. *There is a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$, such that for an arbitrary $A_1, B, A_2 \in \mathcal{C}^*$ in normal form and $\langle A_1 B A_2 \rangle_2 = A'_1 B' A'_2$ where $A'_i, 1 \leq i \leq 2$, is the residual of A_i and B' is the residual of B , we have $|B'| \leq p(|\langle A_1 B A_2 \rangle|)$.*

Proof. We may assume that $\text{card}(\mathcal{C}(B)) \geq 2$ and by Proposition 2 we know that $|B'| \leq 2 \cdot |\langle B' \rangle|^2$, which is of course less or equal to $2 \cdot |\langle A'_1 \rangle \langle B' \rangle|^2$. Since $\langle \langle A'_1 \rangle \langle B' \rangle \rangle_2 = \langle A'_1 \rangle \langle B' \rangle$ by Lemma 8, we can use Proposition 3 and we get that $2 \cdot |\langle A'_1 \rangle \langle B' \rangle|^2 \leq 2 \cdot |\langle A'_1 B' \rangle|^4$, which is again less or equal to $2 \cdot |\langle A'_1 B' \rangle \langle A'_2 \rangle|^4$. Analogously we have that $2 \cdot |\langle A'_1 B' \rangle \langle A'_2 \rangle|^4 \leq 2 \cdot |\langle A'_1 B' A'_2 \rangle|^8$. Thus we have $|B'| \leq p(|\langle A_1 B A_2 \rangle|)$ for the polynomial $p(n) = 2 \cdot n^8$. $\square$

Proposition 5. *Let p be a polynomial that satisfies the condition from Proposition 4. If a stuttering pattern equation system $\{X_1 = A_1, \dots, X_n = A_n\}$ is satisfiable then there exists a solution α with $\text{size}(\alpha) \leq \sum_{i=1}^n |X_i| \cdot p(|A_i|)$.*

Proof. Of course, we can assume that all A_i 's are in normal form. Let α be a solution of the stuttering pattern equation system $\{X_1 = A_1, \dots, X_n = A_n\}$ which minimizes both $\text{size}(\alpha)$ and the number of variables x such that $|\alpha(x)| = \text{size}(\alpha)$. Assume for the moment that there is some x such that $\text{size}(\alpha) = |\alpha(x)| > \sum_{i=1}^n |X_i| p(|A_i|)$. We may assume that $\alpha(x)$ is in normal form, otherwise we have a smaller solution.

We now reduce $\alpha(X_i) \rightarrow_{2m}^* \langle \alpha(X_i) \rangle_2$. If we look at an arbitrary residual B' of an occurrence of $\alpha(x)$ in $\langle \alpha(X_i) \rangle_2$, we see that $|B'| \leq p(|A_i|)$ by Proposition 4. This means that there are at most $\sum_{i=1}^n |X_i| p(|A_i|)$ letter's occurrences in the residuals of all occurrences of $\alpha(x)$ in all $\langle \alpha(X_i) \rangle_2$. By the assumption $|\alpha(x)| > \sum_{i=1}^n |X_i| p(|A_i|)$ we get that there is an occurrence of a letter a in $\alpha(x)$, i.e. $\alpha(x) = u_1 a u_2$, that has been left out from all the occurrences of $\alpha(x)$ by the rule $\rightarrow_{2m}$. We can erase this occurrence of the letter a from $\alpha(x)$ and we get a smaller solution β s.t. $\beta(y) = \alpha(y)$ for $y \neq x$ and $\beta(x) = u_1 u_2$. The homomorphism β is indeed a solution since $\alpha(X_i) \rightarrow_{2l}^* \beta(X_i)$. This is a contradiction because we have found a smaller solution. $\square$

Corollary 1. *The PATTERN-EQUATION problem is in NP.*

5 NP-Hardness of the PATTERN-EQUATION Problem

In this section we show that the PATTERN-EQUATION problem is NP-hard. We use a reduction from the NP-complete problem 3-SAT (see [16]).

Proposition 6. *The PATTERN-EQUATION problem is NP-hard.*

Proof. Suppose we have an instance of 3-SAT, i.e. $C \equiv C_1 \wedge C_2 \wedge \dots \wedge C_n$ is a conjunction of clauses and each clause C_i , $1 \leq i \leq n$, is of the form $l_1 \vee l_2 \vee l_3$ where l_j , $1 \leq j \leq 3$, is a literal (l_j is a variable from the set Var , possibly negated – we call it positive resp. negative literal). A valuation is a mapping $v : Var \rightarrow \{T, F\}$. This valuation extends naturally to C and we say that C is satisfiable if and only if there exists a valuation v such that $v(C) = T$.

We construct a stuttering pattern equation system such that the system is satisfiable if and only if C is satisfiable. The system will consist of the following sets of equations (1) – (6) and $\mathcal{C} = \{a, b, c\}$, $\mathcal{V} = \{x, s_1^x, t_1^x, s_2^x, t_2^x \mid x \in Var \cup \overline{Var}\} \cup \{y_a, y_b, y_c\}$ where $\overline{Var} = \{\bar{x} \mid x \in Var\}$ is a disjoint copy of Var . For the constants a, b and c there are three equations

$$y_a = a, \quad y_b = b, \quad y_c = c. \quad (1)$$

We define $\widetilde{x} = x$ if x is a positive literal, $\widetilde{\neg x} = \bar{x}$ if $\neg x$ is a negative literal and for all clauses $C_i \equiv l_1 \vee l_2 \vee l_3$ we have the equation

$$y_a \widetilde{l_1} \widetilde{l_2} \widetilde{l_3} y_a = aba \quad (2)$$

for each $x \in Var$ we add the equations

$$y_b x \bar{x} y_b = bab \quad (3)$$

$$y_a x \bar{x} y_a = aba \quad (4)$$

and finally for each $x \in Var \cup \overline{Var}$ we have the following equations:

$$s_1^x x t_1^x = acb, \quad s_1^x y_c = ac \quad (5)$$

$$s_2^x x t_2^x = bca, \quad s_2^x y_c = bc. \quad (6)$$

The intuition behind the construction is following. If a variable x is true then $x = b$ and if x is false then $x = a$. The second equation ensures that at least one literal in each clause is true and the other equations imply consistency, i.e. a literal and its negation cannot be both true (false). In particular, the equation (3) means that at least one of x and $\bar{x}$ contains a . Similarly for b and (4). The last two equations make sure that a variable $x \in Var \cup \overline{Var}$ cannot contain both a and b .

Suppose that C is satisfiable, i.e. there is a valuation v such that $v(C) = T$. Then we can easily define a solution of our system.

Let us suppose that α is an arbitrary solution of our system and we find a valuation that satisfies C . The equation (3) implies that $C(\alpha(x)) \subseteq \{a, b\}$ for all $x \in Var \cup \overline{Var}$. We will conclude that it is not possible that $C(\alpha(x)) = \{a, b\}$.

Suppose that it is the case and using the equations (5) we get that $\alpha(x)$ does not begin with the constant a . For the moment assume that $\alpha(x)$ begins with a . We have $ac = 0(acb) \sim 0(\alpha(s_1^x x t_1^x))$ and from (1) and (5) we get that $a \in C(\alpha(s_1^x)) \subseteq \{a, c\}$. If $C(\alpha(s_1^x)) = \{a\}$ then $C(0(\alpha(s_1^x x t_1^x))) = \{a, b\}$ whereas $C(0(acb)) = \{a, c\}$, which is a contradiction. Otherwise we have $C(\alpha(s_1^x)) = \{a, c\}$ and we get $0(\alpha(s_1^x x t_1^x)) \sim aca \not\sim ac = 0(acb)$.

By the similar arguments and using the equations (6) we get that $\alpha(x)$ does not begin with the constant b . This yields that there are just three possibilities for $\alpha(x)$, namely $\alpha(x) \sim a$, $\alpha(x) \sim b$ or $\alpha(x) = \epsilon$.

By the equations (3) and (1) we know that for all $x \in Var$ at least $\alpha(x) \sim a$ or $\alpha(\bar{x}) \sim a$. The equation (4) implies that either $\alpha(x) \sim b$ or $\alpha(\bar{x}) \sim b$. Similarly for each clause, the equation (2) with (1) gives that there is j , $1 \leq j \leq 3$, such that $\alpha(\tilde{l}_j) \sim b$. Let us finally define the valuation v as $v(x) = T$ if $\alpha(x) \sim b$ and $v(x) = F$ if $\alpha(x) \sim a$ for each $x \in Var$. The valuation is consistent and it holds that $v(C) = T$. $\square$

It is not difficult to see that the same reduction as above would also work for the NON-SINGULAR-PATTERN-EQUATION problem, which is consequently also NP-hard. We can now formulate the main result of this paper.

Theorem 1. *PATTERN-EQUATION and NON-SINGULAR-PATTERN-EQUATION problems are NP-complete.*

As an immediate corollary (using Remark 1), we get the following result.

Corollary 2. *AI-matching with only one associative and idempotent function symbol is NP-complete.*

Acknowledgements. We would like to thank Ivana Černá and Michal Kunc for their comments and suggestions.

References

- [1] Baader F.: The Theory of Idempotent Semigroups is of Unification Type Zero, *J. of Automated Reasoning* **2** (1986) 283–286.
- [2] Baader F.: Unification in Varieties of Idempotent Semigroups, *Semigroup Forum* **36** (1987) 127–145.
- [3] Baader F., Schulz K.U.: Unification in the Union of Disjoint Equational Theories: Combining Decision Procedures, *J. Symbolic Computation* **21** (1996) 211–243.
- [4] Baader F., Siekmann J.H.: *Unification Theory*, *Handbook of Logic in Artificial Intelligence and Logic Programming* (1993) Oxford University Press.
- [5] Book R., Otto F.: *String-Rewriting Systems* (1993) Springer-Verlag.
- [6] Černá I., Klíma O., Srba J.: Pattern Equations and Equations with Stuttering, In *Proceedings of SOFSEM'99, the 26th Seminar on Current Trends in Theory and Practice of Informatics* (1999) 369–378, Springer-Verlag.
- [7] Green J.A., Rees D.: On semigroups in which $x^r = x$, *Proc. Camb. Phil. Soc.* **48** (1952) 35–40.
- [8] Klíma O., Srba J.: Matching Modulo Associativity and Idempotency is NP-complete, Technical report RS-00-13, BRICS, Aarhus University (2000).
- [9] Kaďourek J., Polák L.: On free semigroups satisfying $x^r = x$, *Simon Stevin* **64**, No.1 (1990) 3–19.
- [10] Kapur D., Narendran P.: NP-completeness of the Set Unification and Matching Problems, In *Proceedings of CADE'86*, Springer LNCS volume **230** (1986) 489–495, Springer-Verlag.
- [11] Kopeček I.: Automatic Segmentation into Syllable Segments, *Proc. of First International Conference on Language Resources and Evaluation* (1998) 1275–1279.
- [12] Kopeček I., Pala K.: Prosody Modelling for Syllable-Based Speech Synthesis, *Proceedings of the IASTED International Conference on Artificial Intelligence and Soft Computing*, Cancun (1998) 134–137.
- [13] Lothaire M.: *Algebraic Combinatorics on Words*, Preliminary version available at <http://www-igm.univ-mlv.fr/~berstel/Lothaire/index.html>
- [14] Lothaire, M.: *Combinatorics on Words*, Volume **17** of *Encyclopedia of Mathematics and its Applications* (1983) Addison-Wesley.
- [15] Makanin, G. S.: The Problem of Solvability of Equations in a Free Semigroup, *Mat. Sbornik* **103(2)** (1977) 147–236. (In Russian) English translation in: *Math. USSR Sbornik* **32** (1977) 129–198.
- [16] Papadimitriou, C.H.: *Computational Complexity*, Addison-Wesley Publishing Company (1994), Reading, Mass.
- [17] Perrin D.: Equations in Words, In H. Ait-Kaci and M. Nivat, editors, *Resolution of Equations in Algebraic Structures*, Vol. **2** (1989) 275–298, Academic Press.

- [18] Schulz, K. U.: Makanin's Algorithm for Word Equations: Two Improvements and a Generalization, In Schulz, K.-U. (Ed.), *Proceedings of Word Equations and Related Topics*, 1st International Workshop, IWW-ERT'90, Tübingen, Germany, Vol. **572** of LNCS (1992) 85–150, Berlin-Heidelberg-New York, Springer-Verlag.
- [19] Schmidt-Schauss M.: Unification under Associativity and Idempotence is of Type Nullary, *J. of Automated Reasoning* **2** (1986) 277–281.
- [20] Siekmann J., Szabó P.: A Noetherian and Confluent Rewrite System for Idempotent Semigroups, *Semigroup Forum* **25** (1982).

On NP-Partitions over Posets with an Application to Reducing the Set of Solutions of NP Problems

Sven Kosub

Theoretische Informatik, Julius-Maximilians-Universität Würzburg
Am Hubland, D-97074 Würzburg, Germany
kosub@informatik.uni-wuerzburg.de

Abstract. The boolean hierarchy of k -partitions over NP for $k \geq 2$ was introduced as a generalization of the well-known boolean hierarchy of sets. The classes of this hierarchy are exactly those classes of NP-partitions which are generated by finite labeled lattices. We refine the boolean hierarchy of NP-partitions by considering partition classes which are generated by finite labeled posets. Since we cannot prove it absolutely, we collect evidence for this refined boolean hierarchy to be strict. We give an exhaustive answer to the question which relativizable inclusions between partition classes can occur depending on the relation between their defining posets. The study of the refined boolean hierarchy is closely related to the issue of whether one can reduce the number of solutions of NP problems. For finite cardinality types, assuming the extended boolean hierarchy of k -partitions over NP is strict, we get a complete characterization when such solution reductions are possible.

1 Introduction

Complexity theory usually investigates partitions into two parts (sets) or into infinitely many parts (functions). Starting a study of intermediate partitions into a finite number of parts (say k -partitions), Wagner and the present author [9] introduced the boolean hierarchy $BH_k(NP)$ of k -partitions over NP for $k \geq 2$ as a generalization of the well-studied boolean hierarchy of NP sets. Whereas the latter hierarchy has a very clear structure with no more than two incomparable classes, for $k \geq 3$ the situation turned out to be much more opaque: Already the hierarchy $BH_3(NP)$ does not have bounded width with respect to set inclusion unless the polynomial-time hierarchy collapses.

The hierarchy $BH_k(NP)$ is defined to consist of all classes $NP(f)$ for functions f mapping the set $\{1, 2\}^m$ with $m \geq 1$ to $\{1, 2, \dots, k\}$, that are given in the following way: Let the characteristic function of a k -partition $A = (A_1, \dots, A_k)$ be defined as $c_A(x) = i$ if and only if x is in A_i . (A set B is to be identified with the 2-partition $(B, \bar{B})$). For a function f and NP sets $B_1, \dots, B_m$, the k -partition $A = f(B_1, \dots, B_m)$ is defined by $c_A(x) = f(c_{B_1}(x), \dots, c_{B_m}(x))$. Then, a class $NP(f)$ is given as $NP(f) =_{\text{def}} \{f(B_1, \dots, B_m) \mid B_1, \dots, B_m \in NP\}$. The boolean hierarchy of sets appears as the case $k = 2$.

It has been shown that $\text{BH}_k(\text{NP})$ coincides with the family of all partition classes $\text{NP}(G, f)$ defined by arbitrary finite lattices G with labeling functions $f : G \rightarrow \{1, 2, \dots, k\}$, henceforth referred to as k -lattices. This approach is very useful since frequently k -lattices yield smaller descriptions of classes in $\text{BH}_k(\text{NP})$ than functions. The structure inside $\text{BH}_k(\text{NP})$ is induced by a “simpler-than” relation $\leq$ on such k -lattices: (G, f) is simpler than (G', f') iff there is a mapping φ from G to G' that preserves order and labels. One can prove that this relation translates to partition classes, that is, if (G, f) is simpler than (G', f') then $\text{NP}(G, f) \subseteq \text{NP}(G', f')$. For several good reasons, it has been conjectured that the converse direction is also true. More specifically, the conjecture is that $\text{BH}_k(\text{NP})$ is strict with respect to $\leq$, that is, $\text{NP}(G, f) \subsetneq \text{NP}(G', f')$ whenever (G, f) is not simpler than (G', f') , unless the polynomial-time hierarchy collapses. Unfortunately, this so-called Embedding Conjecture [9] is unproven up to now.

In this paper, we extend the work presented in [9] to the most general case of finite labeled posets (k -posets) by a modification of the lattice approach. In Sect. 3 this will lead us to the refined boolean hierarchy $\text{RBH}_k(\text{NP})$ of k -partition over NP which can be built according to the same simpler-than relation as for lattices. Thus, $\text{BH}_k(\text{NP})$ appears as a subhierarchy of $\text{RBH}_k(\text{NP})$. In Sect. 4 we will describe an alternative approach to $\text{RBH}_k(\text{NP})$ in terms of partial finite functions. We will obtain a correlation in $\text{RBH}_k(\text{NP})$ between posets and partial functions that corresponds perfectly with the correlation between lattices and total functions as on the side of the boolean hierarchy of NP -partitions.

Using this approach, we are able to prove our most striking theorem regarding the strictness of the refined boolean hierarchy of NP -partitions with respect to $\leq$: For all k -posets (G, f) and (G', f') , we have that (G, f) is simpler than (G', f') if and only if *relativizably* $\text{NP}(G, f) \subseteq \text{NP}(G', f')$. This theorem, stated in Sect. 5, underlines the fundamental nature of the relation $\leq$ in investigating boolean hierarchies over NP . For instance, as an immediate consequence we get a complete characterization for the case of k -lattices and thus of the boolean hierarchy $\text{BH}_k(\text{NP})$ providing new and, due to its broad scope, strong evidence for the truth of the above mentioned Embedding Conjecture. Of course the status of relativizations might be considered as somewhat questionable, but our theorem shows that a (unrelativized) counterexample to the strictness of $\text{RBH}_k(\text{NP})$ (and in particular to $\text{BH}_k(\text{NP})$) with respect to $\leq$ must use the very rare non-relativizable proof techniques, hence would be very surprising.

Exploring the refined version of the boolean hierarchy of NP -partitions is interesting since its structural richness turns out to be useful for proving facts about related notions concerning NP . For instance, our main motivation for examining $\text{RBH}_k(\text{NP})$ is its close relation to the issue of whether one can reduce the set of solutions of NP problems. Building on previous work in [1,2,6,11,10], this issue has taken its ultimate shape in a work of Hemaspaandra, Ogihara, and Wechsung [7].

For $A \subseteq \mathbb{N}_+$, let NP_AV consist of all possibly partial, possibly multivalued functions f computed by an NP machine so that $f(x)$ is the set of all outputs

(“solutions”) on accepting paths and the number of outputs lies in $\{0\} \cup A$. The study of this notion is motivated by considering A as a computing resource that strongly influences the computing power of NP machines (for a more detailed discussion see the introductory sections in [7,10]).

Multivalued functions are adequately compared by refinements. A multivalued function f is a refinement of a multivalued function f' if the domains of both functions are equal and $f(x) \subseteq f'(x)$ for all x . This notion induces a relation $\subseteq_c$ on function classes: $\mathcal{F} \subseteq_c \mathcal{F}'$ if each function in $\mathcal{F}$ has a refinement in $\mathcal{F}'$.

Refinements are capturing solution reductions. As an example, one could ask whether it is possible to construct an NP machine that always bounds, for a given satisfiable propositional formula, the cardinality of its output set of satisfying assignments to one. Such a machine reduces the set of solutions of the satisfiability problem to a singleton. The question for the existence of this machine translates to the more formal question “ $\text{NP}_{\mathbb{N}_+} V \subseteq_c \text{NP}_{\{1\}} V$?” for which Hemaspaandra, Naik, Ogihara, and Selman [6] have shown that the answer is negative if the polynomial-time hierarchy does not collapse.

As natural the challenge is: “Completely characterize, perhaps under some complexity-theoretic assumptions, the sets $A \subseteq \mathbb{N}_+$ and $B \subseteq \mathbb{N}_+$ such that $\text{NP}_A V \subseteq_c \text{NP}_B V$ ” [7]. And Hemaspaandra, Ogihara, and Wechsung gave furthermore a sufficient condition for refinements (the “narrowing-gap condition”) in the case of finite sets, and they pronounced the conjecture, supported with strong theorems, that this condition is also necessary unless the polynomial-time hierarchy collapses. In Sect. 6 we prove that in fact, the narrowing-gap condition for sets A and B is sufficient and necessary for $\text{NP}_A V \subseteq_c \text{NP}_B V$ unless for a certain k , some classes in $\text{RBH}_k(\text{NP})$ unexpectedly collapse.

A full version of this paper, including all proofs of all claims, is available as Technical Report No. 257 of the Institut für Informatik, Julius-Maximilians-Universität Würzburg.

2 Preliminaries

For classes $\mathcal{K}$ and $\mathcal{K}'$ of subsets of a set M , define $\text{co}\mathcal{K} = \{\bar{L} \mid L \in \mathcal{K}\}$, $\mathcal{K} \wedge \mathcal{K}' =_{\text{def}} \{A \cap B \mid A \in \mathcal{K}, B \in \mathcal{K}'\}$, and $\mathcal{K} \oplus \mathcal{K}' =_{\text{def}} \{A \Delta B \mid A \in \mathcal{K}, B \in \mathcal{K}'\}$ where $A \Delta B$ denotes the symmetric difference of A and B . For a set A , $\|A\|$ denotes its cardinality.

The classes $\mathcal{K}(i)$ and $\text{co}\mathcal{K}(i)$ defined by $\mathcal{K}(1) =_{\text{def}} \mathcal{K}$ and $\mathcal{K}(i+1) =_{\text{def}} \mathcal{K}(i) \oplus \mathcal{K}$ for $i \geq 1$ build the *boolean hierarchy over $\mathcal{K}$* that has many equivalent definitions (see [12,4,8,3] or the case $k = 2$ in Definition 1). The class $\mathbf{BC}(\mathcal{K})$ is the boolean closure of $\mathcal{K}$, that is the smallest class which contains $\mathcal{K}$ and which is closed under intersection, union, and complementation.

Let us make some notational conventions about partitions. For any set M , a k -tuple $A = (A_1, \dots, A_k)$ is a *k-partition of M* iff $A_1 \cup A_2 \cup \dots \cup A_k = M$ and $A_i \cap A_j = \emptyset$ if $i \neq j$. The set A_i is called the *i-th component of A* . Let $c_A : M \rightarrow \{1, 2, \dots, k\}$ be the characteristic function of a k -partition $A = (A_1, \dots, A_k)$ of

M , i.e., $c_A(x) = i$ if and only if $x \in A_i$. For $\mathcal{K}_1, \dots, \mathcal{K}_k \subseteq \mathcal{P}(M)$, we define

$$(\mathcal{K}_1, \dots, \mathcal{K}_k) =_{\text{def}} \{A \mid A \text{ is a } k\text{-partition of } M \text{ and } A_i \in \mathcal{K}_i \text{ for } 1 \leq i \leq k\}$$

and for $1 \leq i \leq k$,

$$(\mathcal{K}_1, \dots, \mathcal{K}_{i-1}, \cdot, \mathcal{K}_{i+1}, \dots, \mathcal{K}_k) =_{\text{def}} (\mathcal{K}_1, \dots, \mathcal{K}_{i-1}, \mathcal{P}(M), \mathcal{K}_{i+1}, \dots, \mathcal{K}_k).$$

For a class $\mathcal{K}$ of k -partitions, let $\mathcal{K}_i =_{\text{def}} \{A_i \mid A \in \mathcal{K}\}$ be the i -th projection of $\mathcal{K}$. Obviously, $\mathcal{K} \subseteq (\mathcal{K}_1, \dots, \mathcal{K}_k)$. In the forthcoming we identify a set A with the 2-partition $(A, \bar{A})$, and we identify a class $\mathcal{K}$ of sets with the class $(\mathcal{K}, \text{co}\mathcal{K}) = (\mathcal{K}, \cdot) = (\cdot, \text{co}\mathcal{K})$ of 2-partitions.

We recall the definition of the boolean hierarchy of k -partitions over $\mathcal{K}$ for $k \geq 2$ as introduced in [9].

Definition 1. Let $k \geq 2$.

1. For $f : \{1, 2\}^m \rightarrow \{1, \dots, k\}$ and for sets $B_1, \dots, B_m \in \mathcal{K}$, the k -partition $f(B_1, \dots, B_m)$ is defined by $c_{f(B_1, \dots, B_m)}(x) = f(c_{B_1}(x), \dots, c_{B_m}(x))$.
2. For $f : \{1, 2\}^m \rightarrow \{1, \dots, k\}$, the class of k -partitions over $\mathcal{K}$ defined by f is given by the class $\mathcal{K}(f) =_{\text{def}} \{f(B_1, \dots, B_m) \mid B_1, \dots, B_m \in \mathcal{K}\}$.
3. The family $\text{BH}_k(\mathcal{K}) =_{\text{def}} \{\mathcal{K}(f) \mid f : \{1, 2\}^m \rightarrow \{1, 2, \dots, k\}\}$ is the boolean hierarchy of k -partitions over $\mathcal{K}$.
4. $\text{BC}_k(\mathcal{K}) =_{\text{def}} \bigcup \text{BH}_k(\mathcal{K})$.

We mention some notions from lattice theory and order theory (see e.g., [5]). A finite poset $(G, \leq)$ is a lattice if for all $x, y \in G$ there exist exactly one maximal element $z \in G$ such that $z \leq x$ and $z \leq y$ (which will be denoted by $x \wedge y$), and exactly one minimal element $z \in G$ such that $z \geq x$ and $z \geq y$ (which will be denoted by $x \vee y$). For a lattice G we denote by 1_G the unique element greater than or equal to all $x \in G$.

For symbols $a_1, a_2, \dots, a_m$ from a finite alphabet Σ , we identify the m -tuple $(a_1, a_2, \dots, a_m)$ with the word $a_1 a_2 \dots a_m \in \Sigma^m$. We denote the length of a word w by $|w|$ and for a letter $a \in \Sigma$, the number of occurrence of the letter a in w by $|w|_a$. If there is an order $\leq$ on Σ , we assume Σ^m to be partially ordered by the vector-ordering, that is $a_1 a_2 \dots a_m \leq b_1 b_2 \dots b_m$ if and only if $a_i \leq b_i$ for all $i \in \{1, 2, \dots, m\}$. The lexicographical ordering on Σ^m is denoted by $\leq_{\text{lex}}$.

The domain of any function f is denoted by D_f . For a multivalued function (or set function) f we define $D_f =_{\text{def}} \{x \mid f(x) \neq \emptyset\}$.

3 Partition Classes Defined by Posets

In [9] the boolean hierarchy of k -partitions over a set class $\mathcal{K}$ was precisely characterized by partition classes defined by finite labeled lattices. In this section we will generalize this approach to the cases of arbitrary posets rather than lattices. In doing so we follow completely the line offered in [9].

Let $\mathcal{K}$ be a class of subsets of M such that $\emptyset, M \in \mathcal{K}$ and $\mathcal{K}$ is closed under union and intersection.

Definition 2. Let G be a poset.

1. A mapping $S : G \rightarrow \mathcal{K}$ is said to be a $\mathcal{K}$ -homomorphism on G if and only if $\bigcup_{a \in G} S(a) = M$ and $S(a) \cap S(b) = \bigcup_{c \leq a, c \leq b} S(c)$ for all $a, b \in G$.
2. For any $\mathcal{K}$ -homomorphism S on G , let $T_S(a) =_{\text{def}} S(a) \setminus \bigcup_{b < a} S(b)$ for all $a \in G$.

Note that in the case of a lattice G the above notion of a $\mathcal{K}$ -homomorphism on G coincides with the one for lattices [9]: $S(1_G) = M$ and $S(a) \cap S(b) = S(a \wedge b)$.

Lemma 1. Let G be a poset, and let S be a $\mathcal{K}$ -homomorphism on G .

1. $T_S(a) \in \mathcal{K} \wedge \text{co}\mathcal{K}$ for every $a \in G$.
2. If $a \leq b$ then $S(a) \subseteq S(b)$ for every $a, b \in G$.
3. $S(a) = \bigcup_{b \leq a} T_S(b)$ for every $a \in G$.
4. The set of all $T_S(a)$ for $a \in G$ yields a partition of M .

Any pair (G, f) of an arbitrary finite poset G and a function $f : G \rightarrow \{1, \dots, k\}$ is called a k -poset. A k -poset which is a lattice (boolean lattice, chain, etc.) is called a k -lattice (boolean k -lattice, k -chain, etc.).

Lemma 1 provides that the following definitions are sound.

Definition 3. Let $k \geq 2$.

1. For a k -poset (G, f) and a $\mathcal{K}$ -homomorphism S on G let $(G, f, S) =_{\text{def}} (\bigcup_{f(a)=1} T_S(a), \dots, \bigcup_{f(a)=k} T_S(a))$ be the k -partition defined by (G, f) and S .
2. Let $\mathcal{K}(G, f) =_{\text{def}} \{(G, f, S) \mid S \text{ is } \mathcal{K}\text{-homomorphism on } G\}$ be the class of k -partitions defined by the k -poset (G, f) .
3. The family $\text{RBH}_k(\mathcal{K}) =_{\text{def}} \{\mathcal{K}(G, f) \mid (G, f) \text{ is a } k\text{-poset}\}$ is the refined boolean hierarchy of k -partitions over $\mathcal{K}$.

Figure 1 shows some examples of partition classes in $\text{RBH}_3(\mathcal{K})$ together with their defining 3-posets. We should mention here that in general, even if it seems so, not all partition classes over posets can exactly be described in their components by classes of the boolean hierarchy of sets.

The following propositions relate the refined boolean hierarchy to the boolean hierarchy.

Proposition 1. $\text{BH}_k(\mathcal{K}) \subseteq \text{RBH}_k(\mathcal{K})$ for every $k \geq 2$.

Proposition 2. Let $\mathcal{K}$ be not closed under complementation, and let $k \geq 2$. Then there exists a partition class in $\text{RBH}_k(\mathcal{K})$ which does not belong to $\text{BH}_k(\mathcal{K})$.

Proposition 3. Let (G, f) be a k -poset with $f : G \rightarrow \{1, 2, \dots, k\}$ surjective.

1. $(\mathcal{K}, \mathcal{K}, \dots, \mathcal{K}) \subseteq \mathcal{K}(G, f) \subseteq (\mathbf{BC}(\mathcal{K}), \mathbf{BC}(\mathcal{K}), \dots, \mathbf{BC}(\mathcal{K})) = \mathbf{BC}_k(\mathcal{K})$.
2. If $\mathcal{K}$ is closed under complementation then $\mathcal{K}(G, f) = (\mathcal{K}, \mathcal{K}, \dots, \mathcal{K})$.

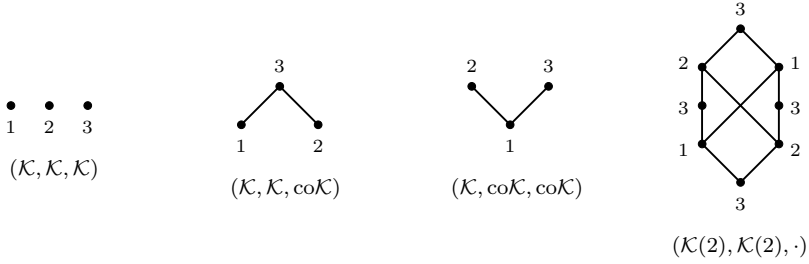


Fig. 1. Some partition classes in $\text{RBH}_3(\mathcal{K})$ and their defining 3-posets.

Given two k -posets (G, f) and (G', f') it would be very useful to have a criterion to decide whether $\mathcal{K}(G, f) \subseteq \mathcal{K}(G', f')$. The following lemma provides a sufficient condition.

For k -posets (G, f) and (G', f') we write $(G, f) \leq (G', f')$ if and only if there is a monotonic mapping $\varphi : G \rightarrow G'$ such that $f(x) = f'(\varphi(x))$ for every $x \in G$.

Lemma 2. (Embedding Lemma.) *Let (G, f) and (G', f') be k -posets. Then, if $(G, f) \leq (G', f')$, then $\mathcal{K}(G, f) \subseteq \mathcal{K}(G', f')$.*

4 An Equivalent Approach

In Section 3 it has been described how to define classes of k -partitions by posets. This has generalized the case of lattices. In this section we will make another generalization of the lattice approach which results in the same classes of k -partitions as obtained in the poset approach.

Let $\mathcal{K}$ be a class of subsets of M such that $\emptyset, M \in \mathcal{K}$ and $\mathcal{K}$ is closed under union and intersection.

Definition 4. *Let G be a lattice and let $H \subset G$.*

1. *A mapping $S : G \rightarrow \mathcal{K}$ is said to be a $\mathcal{K}$ -homomorphism on (G, H) if and only if $S(1_G) = M$, $S(a \wedge b) = S(a) \cap S(b)$ for all $a, b \in G$, and $S(a) = \bigcup_{b < a} S(b)$ for all $a \in H$.*
2. *For any $\mathcal{K}$ -homomorphism S on (G, H) , let $T_S(a) =_{\text{def}} S(a) \setminus \bigcup_{b < a} S(b)$ for all $a \in G$.*

As immediate consequences of this definition we obtain

Proposition 4. *Let S be a $\mathcal{K}$ -homomorphism on (G, H) for a lattice G and $H \subset G$.*

1. *S is a $\mathcal{K}$ -homomorphism on G .*
2. *$T_S(a) = \emptyset$ for all $a \in H$.*

Proposition 5. *Let S be a $\mathcal{K}$ -homomorphism on a lattice G and let $H \subset G$. If $T_S(a) = \emptyset$ for all $a \in H$, then S is a $\mathcal{K}$ -homomorphism on (G, H) .*

Consequently we can apply all definitions made for $\mathcal{K}$ -homomorphisms on G also to $\mathcal{K}$ -homomorphisms on (G, H) . So we are able to define classes of k -partitions.

Definition 5. For a k -lattice (G, f) and a set $H \subset G$ define

$$\mathcal{K}(G, H, f) =_{\text{def}} \{(G, f, S) \mid S \text{ is a } \mathcal{K}\text{-homomorphism on } (G, H)\}.$$

It turns out that every such class $\mathcal{K}(G, H, f)$ is also of the form $\mathcal{K}(G, f)$ where (G, f) is a k -poset and vice versa. Furthermore, we can equivalently consider boolean k -lattices (G, f) and $H \subset G$.

Theorem 1.

$$\begin{aligned} \text{RBH}_k(\mathcal{K}) &= \{\mathcal{K}(G, H, f) \mid (G, f) \text{ is a boolean } k\text{-lattice and } H \subset G\} \\ &= \{\mathcal{K}(G, H, f) \mid (G, f) \text{ is a } k\text{-lattice and } H \subset G\}. \end{aligned}$$

Theorem 1 can be restated in terms of partial functions giving a similar correspondence between posets and partial functions as known for lattices and total functions.

Theorem 2. Let (G, f) be a k -poset. There is a partial function $f' : \{1, 2\}^m \rightarrow \{1, \dots, k\}$ such that a k -partition L belongs to $\mathcal{K}(G, f)$ if and only if there exist sets $B_1, \dots, B_m \in \mathcal{K}$ with $(c_{B_1}(x), \dots, c_{B_m}(x)) \in D_{f'}$ and $c_L(x) = f'(c_{B_1}(x), \dots, c_{B_m}(x))$ for all $x \in M$.

5 On the Strictness of $\text{RBH}_k(\text{NP})$

In Sect. 3 we have defined partition classes over posets and we have identified a relation inducing a sufficient criterion for inclusions between partition classes. In this section, focusing our attention to the special case $\mathcal{K} = \text{NP}$, we will give some reasons why we are convinced of the sufficient criterion to be also necessary or synonymously, that $\text{RBH}_k(\text{NP})$ is strict, i.e., $\text{NP}(G, f) \not\subseteq \text{NP}(G', f')$ whenever $(G, f) \not\leq (G', f')$.

Clearly, all results from [9] providing evidence for the strictness of $\text{BH}_k(\text{NP})$ are also arguments for the strictness of $\text{RBH}_k(\text{NP})$. Moreover, some proofs carry over to the case of posets. The following theorem is an example.

A repetition-free k -chain is one where neighbored elements have different labels.

Theorem 3. Assume the polynomial-time hierarchy does not collapse. Let (G, f) and (G', f') be k -posets. If $\text{NP}(G, f) \subseteq \text{NP}(G', f')$, then each repetition-free k -subchain of (G, f) occurs as a k -subchain of (G', f') .

Building on results like this, the Embedding Conjecture (for lattices) was posed stating that $\text{BH}_k(\text{NP})$ is strict unless the polynomial-time hierarchy collapses. Though we are also convinced of the strictness of $\text{RBH}_k(\text{NP})$, we must be warned to literally take over this conjecture for posets.



Fig. 2. Critical 3-posets.

Proposition 6. *Let (G, f) and (G', f') be the 3-posets in Fig. 2. Then $(G, f) \not\leq (G', f')$ but there exists an oracle where the polynomial-time hierarchy does not collapse to any level and $\text{NP}(G, f) = \text{NP}(G', f')$.*

Does this oracle also give evidence against the Embedding Conjecture for lattices? We feel that this is not true since the counterexample strongly depends on the structural weakness of posets. Moreover, the oracle result affects rather the seemingly insufficient assumption of a strict polynomial-time hierarchy than the correspondence between k -posets and k -partition classes. For instance, under the widely believed assumption that $\text{UP} \not\subseteq \text{coNP}$, the counterexample fails.

Proposition 7. *Let (G, f) and (G', f') be the 3-posets in Fig. 2. If $\text{NP}(G, f) = \text{NP}(G', f')$ then $\text{UP} \subseteq \text{coNP}$.*

Making use of our alternative characterization of classes in $\text{RBH}_k(\text{NP})$, in the following we state that we cannot collapse unexpected classes in $\text{RBH}_k(\text{NP})$ using relativizable proof techniques. On the one hand, certainly, this is not such a strong statement as Theorem 3 for the classes in its scope, but on the other side it is a statement about *all* classes of $\text{RBH}_k(\text{NP})$, hence our most striking result regarding the strictness of $\text{RBH}_k(\text{NP})$.

Theorem 4. *Let (G, f) and (G', f') be k -posets. Then $(G, f) \leq (G', f')$ if and only if $\text{NP}^C(G, f) \subseteq \text{NP}^C(G', f')$ for every oracle C .*

6 Application

In Sect. 3 and Sect. 4 we established the refined boolean hierarchy of k -partitions and gave alternative characterizations of the classes in it. In the previous section we presented arguments supporting the conjecture that $\text{RBH}_k(\text{NP})$ is strict. In this section we will show that $\text{RBH}_k(\text{NP})$ is closely related to other issues in complexity theory such as the issue of whether one can reduce the set of solutions of NP problems.

The class NPMV introduced by Book, Long, and Selman [1,2] contains all possibly partial, possibly multivalued functions (set functions) f for which there exists a nondeterministic polynomial-time Turing transducer such that $f(x)$ is exactly the set of all outputs (“solutions”) made by the transducer on x on accepting paths. For any $A \subseteq \mathbb{N}_+$, let NP_AV denote the class of all NPMV functions f satisfying for all $x \in \Sigma^*$ that the number of solutions of $f(x)$ is an element of $\{0\} \cup A$.

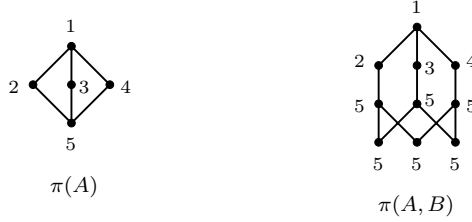


Fig. 3. The 5-posets for the pair (A, B) where $A = \{1, 3\}$ and $B = \{1, 2\}$.

Solution reduction (or output reduction) is captured by the notion of refinements. Let $\mathcal{F}$ and $\mathcal{G}$ be classes of (possibly partial, possibly multivalued) functions. Then we say that $\mathcal{F}$ functions always have $\mathcal{G}$ refinements, in symbols $\mathcal{F} \subseteq_c \mathcal{G}$, iff for all $f \in \mathcal{F}$ there is a $g \in \mathcal{G}$ such that $D_f = D_g$ and $g(x) \subseteq f(x)$ for all $x \in \Sigma^*$.

We focus on the problem of determining for which finite sets $A \subseteq \mathbb{N}_+$ and $B \subseteq \mathbb{N}_+$ it holds that $\text{NP}_A \text{V} \subseteq_c \text{NP}_B \text{V}$. To cope this challenge, the following property has been detected as a promising one.

Definition 6. [7] Let $A, B \subseteq \mathbb{N}_+$ be finite, $A = \{a_1, \dots, a_m\}$ with $a_1 < a_2 < \dots < a_m$. We say that the pair (A, B) satisfies the narrowing-gap condition if and only if $\|A\| = 0$ or there exist $b_1, \dots, b_m \in B$ such that $a_1 - b_1 \geq a_2 - b_2 \geq \dots \geq a_m - b_m \geq 0$.

Narrowing gaps are sufficient for refinements.

Theorem 5. [7] Let $A, B \subseteq \mathbb{N}_+$ be finite. If (A, B) satisfies the narrowing-gap condition, then $\text{NP}_A \text{V} \subseteq_c \text{NP}_B \text{V}$.

And very sophisticated necessary conditions for refinements have been proven assuming the polynomial-time hierarchy is strict (cf. [7,6,11,10]). However, the theorems do not fully match the narrowing-gap condition.

Adopting $\text{RBH}_k(\text{NP})$ to be strict for all $k \geq 2$, we can prove that in fact, the narrowing-gap condition is necessary for refinements. To do that, we define particular posets representing a pair of finite sets. Note that if $\min A < \min B$ then $\text{NP}_A \text{V} \not\subseteq_c \text{NP}_B \text{V}$ unrelativized (and in all relativizations).

Definition 7. Let $A, B \subseteq \mathbb{N}_+$ be finite with $\min B \leq \min A$.

1. Then, $\pi(A)$ denotes the finite labeled poset $((G, \leq), f)$ with
 - $G = \{x \in \{1, 2\}^m \mid |x|_1 \in \{0\} \cup A\}$,
 - $f(x) = \|\{z \in G \mid x \leq_{\text{lex}} z\}\|$ for all $x \in G$.
2. Then, $\pi(A, B)$ denotes the finite labeled poset $((G, \leq), f)$ with
 - $G = \{(x, y) \mid x, y \in \{1, 2\}^m, |x|_1 \in A, |y|_1 \in B, x \leq y\} \cup \{(2^m, 2^m)\}$,
 - $f(x, y) = \|\{z \mid x \leq_{\text{lex}} z \wedge (\exists t)[(z, t) \in G]\}\|$ for all $(x, y) \in G$.

In Fig. 3, posets representing the question of whether $\text{NP}_{\{1,3\}} \text{V} \subseteq_c \text{NP}_{\{1,2\}} \text{V}$ (which is known to imply a collapse of the polynomial-time hierarchy) are drawn. Observe that $\pi(A) \not\subseteq \pi(A, B)$.

Theorem 6. Let $A, B \subseteq \mathbb{N}_+$ be finite with $\min B \leq \min A$. Then the following statements are equivalent.

1. $\text{NP}_A V^D \subseteq_c \text{NP}_B V^D$ for all oracles D .
2. $\text{NP}^D(\pi(A)) \subseteq \text{NP}^D(\pi(A, B))$ for all oracles D .
3. $\pi(A) \leq \pi(A, B)$.
4. (A, B) satisfies the narrowing-gap condition.

Corollary 1. Assume the hierarchy $\text{RBH}_k(\text{NP})$ is strict for all $k \geq 2$. Let $A, B \subseteq \mathbb{N}_+$ be finite. Then (A, B) satisfies the narrowing-gap condition if and only if $\text{NP}_A V \subseteq_c \text{NP}_B V$.

Corollary 2. Let $A, B \subseteq \mathbb{N}_+$ be finite. Then (A, B) satisfies the narrowing-gap condition if and only if $\text{NP}_A V^D \subseteq_c \text{NP}_B V^D$ for all oracles D .

Acknowledgements. I am grateful to Klaus W. Wagner for his permanent suggestions. I also thank Lance Fortnow and Heribert Vollmer for valuable hints.

References

1. R. V. Book, T. J. Long, and A. L. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13:461–487, 1984.
2. R. V. Book, T. J. Long, and A. L. Selman. Qualitative relativizations of complexity classes. *Journal of Computer and System Sciences*, 30(3):395–413, 1985.
3. J.-Y. Cai, T. Gundermann, J. Hartmanis, L. A. Hemachandra, V. Sewelson, K. W. Wagner, and G. Wechsung. The boolean hierarchy I: Structural properties. *SIAM Journal on Computing*, 17(6):1232–1252, 1988.
4. J.-Y. Cai and L. Hemachandra. The Boolean hierarchy: hardware over NP. In *Proceedings 1st Structure in Complexity Theory Conference*, volume 223 of *Lecture Notes in Computer Science*, pages 105–124. Springer-Verlag, 1986.
5. G. Grätzer. *General Lattice Theory*. Akademie-Verlag, Berlin, 1978.
6. L. A. Hemaspaandra, A. V. Naik, M. Ogihara, and A. L. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, 25(4):697–708, 1996.
7. L. A. Hemaspaandra, M. Ogihara, and G. Wechsung. Reducing the number of solutions of NP functions. In *Proceedings 25th Symposium on Mathematical Foundations of Computer Science*. These Proceedings.
8. J. Köbler, U. Schöning, and K. W. Wagner. The difference and truth-table hierarchies for NP. *RAIRO Theoretical Informatics and Applications*, 21(4):419–435, 1987.
9. S. Kosub and K. W. Wagner. The boolean hierarchy of NP-partitions. In *Proceedings 17th Symposium on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 157–168, Berlin, 2000. Springer-Verlag.
10. A. V. Naik, J. D. Rogers, J. S. Royer, and A. L. Selman. A hierarchy based on output multiplicity. *Theoretical Computer Science*, 207(1):131–157, 1998.
11. M. Ogihara. Functions computable with limited access to NP. *Information Processing Letters*, 58(1):35–38, 1996.
12. K. W. Wagner and G. Wechsung. On the boolean closure of NP. Extended abstract as: G. Wechsung. On the boolean closure of NP. *Proceedings 5th International Conference on Fundamentals in Computation Theory*, volume 199 of *Lecture Notes in Computer Science*, pages 485–493, Berlin, 1985.

Algebraic and Uniqueness Properties of Parity Ordered Binary Decision Diagrams and Their Generalization

(Extended Abstract)

Daniel Král'^{*}

Charles University
Malostranské náměstí 25, 118 00 Praha 1
Czech Republic
`kral@atrey.karlin.mff.cuni.cz`

Abstract. Ordered binary decision diagrams (OBDDs) and parity OBDDs are data structures representing Boolean functions. In addition, we study their generalization which we call parity AOBDDs, give their algebraic characterization and compare their minimal size to the size of parity OBDDs.

We prove that the constraint that no arcs test conditions of type $x_i = 0$ does not affect the node-size of parity (A)OBDDs and we give an efficient algorithm for finding such parity (A)OBDDs. We obtain a canonical form for parity OBDDs and discuss similar results for parity AOBDDs.

Algorithms for minimization and transformation to the canonical form for parity OBDDs running in time $O(S^3)$ and space $O(S^2)$ or in time $O(S^3/\log S)$ and space $O(S^3/\log S)$ and an algorithm for minimization of parity AOBDDs running in time $O(nS^3)$ and space $O(nS^2)$ are presented (n is the number of variables, S is the number of vertices).

All the results are extendable to case of shared parity (A)OBDDs — data structures for representation of Boolean function sequences.

1 Introduction

Data structures representing Boolean functions play a key role in formal circuit verification. They are also important as combinatorial structures corresponding to Boolean functions. Once a data structure representing Boolean functions is chosen it should allow compact representation of many important functions and fast implementation of fundamental algorithms (for surveys see [3], [9], [10]).

Graph-based data structures allow to implement algorithms for Boolean function manipulation using graph algorithms. Ordered binary decision diagrams (OBDDs) were introduced in [2]. Parity OBDDs, more powerful modification of OBDDs, were introduced in [5] and further investigated in [8]. Further information may be found in [9]. In addition to parity OBDDs we also consider their

^{*} Supported by GAUK 158/99

new generalization — parity arc-ordered binary decision diagrams (AOBDDs) — for definitions of both parity OBDDs and parity AOBDDs see Section 2.

The size-minimality of used data structures plays an essential role in efficiency of algorithms. The minimal data structure representing the given function can possibly have more different non-isomorphic forms. We look for a canonical form which must be unique; it must also be size-minimal, exist for all Boolean functions and be polynomially computable from any non-canonical form (see [3]). We choose a canonical form for parity OBDDs and present an algorithm for constructing it similar to Waack's algorithm for minimization. The presented algorithm is faster and its last (transform) phase is also simpler since it does not use the Gaussian elimination procedure. The details can be found in [6].

In Section 2 we give definitions of used data structures and introduce notation used in the paper. In Section 3 we study the size-minimality of parity (A)OBDDs. We prove that the constraint that parity BDDs, parity OBDDs and parity AOBDDs do not contain negative arcs, i.e. arcs testing conditions $x_i = 0$, does not affect the node-size of the node-minimal diagram (Theorem 1). We give an efficient algorithm (*Removal of zero arcs*) for finding such parity (A)OBDDs (see Subsection 6.3). The main theorem of Section 3 is the algebraic characterization theorem for parity AOBDDs (Theorem 2). The relationship between minimal parity OBDDs and parity AOBDDs is discussed in Theorem 4.

We define the uniqueness conditions for parity (A)OBDDs and we study properties of diagrams satisfying these conditions in Section 4. Unfortunately, parity AOBDDs representing the same function which satisfy the uniqueness conditions need not to be isomorphic. We prove the canonicity of the representations which satisfy the uniqueness conditions for parity OBDDs (Theorem 10). We give an efficient algorithm (*Unification*) for finding structures which satisfy the uniqueness conditions (see Subsection 6.4). We give a linear-time algorithm for finding the isomorphism between canonical forms (the *PDFS algorithm*). All these results can be extended to case of shared parity (A)OBDDs — see [6].

Waack ([8]) presented the algorithm for node-minimization of parity OBDDs running in time $O(nS^\omega)$ where ω is the exponent of matrix multiplication (currently the best achieved one is 2.376, see [4]). and in space $O(S^2)$ where S is the size of the diagram (the number of its vertices). Löbbing, Sieling and Wegener ([7]) proved that if there exists an algorithm for node-minimization of parity OBDDs running in time $O(t(S))$ then there exists an algorithm for computing the rank of a Boolean $S \times S$ matrix running in time $O(t(S))$ thus we can hardly hope to find a practicle-usable algorithm for node-minimization of parity OBDDs running in time very different from $\Theta(S^3)$. In Section 6, we describe two algorithms for node-minimization of parity OBDDs (for overview of their running times see Table 2). The algorithm running in time $O(S^3)$ does not use any of methods for fast matrix multiplication. The application of Gaussian elimination procedure is completely eliminated from the last (transform) phase of minimization in both algorithms.

2 Definitions and Basic Properties

Let us denote by F_2 the two-element field. We understand the set B_n of Boolean functions of n variables as 2^n -dimensional vector space over F_2 (see also [8]).

A *parity arc-ordered binary decision diagram* with respect to a permutation π of the set $\{1, 2, \dots, n\}$ is an acyclic directed graph with the properties described below. We abbreviate the name of the structure to $\oplus AOBDD$ or to $\pi\text{-}\oplus AOBDD$. There are two special vertices: a *source* and a *sink*. If the function is all-zero then the presence of the sink is not necessary. Arcs from the source are unlabelled; all other arcs are labelled with a pair consisting of a variable x_i and an element of F_2 . We call arcs labelled with zero (one) *negative arcs* (*positive arcs*). Every sequence of variable-indices induced by variables labelled to arcs of any dipath from the source to the sink is strictly π -increasing, i.e. we demand to query the variables in a subsequence of $x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)}$. With an additional constraint that arcs from any vertex have to be labelled with the same variable our definition becomes the definition of *parity OBDDs* (see [8]); in this case we label the vertices instead of the arcs with the variables. If we leave out the variable-ordering constraint the definition of parity OBDDs becomes the definition of *parity BDDs*.

We consider the number of vertices as the size of the $\oplus AOBDD$ as in case of $\oplus OBDDs$ ([8]). The actual storage size of a $\oplus AOBDD$ with the size S belongs both to $\Omega(S)$ and $O(nS^2)$. We write f_B for the function represented by a $\oplus(A)OBDD$ B . The value of $f_B(w_1, w_2, \dots, w_n)$ is 1 iff there is an odd number of dipaths from the source to the sink using only admissible arcs for an assignment w_i to x_i . For a variable-assignment the set of admissible arcs is the set of all unlabelled arcs and arcs for which their labellings are consistent with the assignment. It is possible to extend $\oplus(A)OBDD$ to a *shared parity (A)OBDD* to represent the set of Boolean functions in the same manner as in [8].

Let f_v , the function represented by the vertex v of $\oplus(A)OBDD$, be a function of Boolean variables $x_1, \dots, x_n$ which equals 1 iff there is an odd number of admissible dipaths for the given variable-assignment from v to the sink. Note that if v is the source (the sink) then f_v is the function represented by the $\oplus AOBDD$ (the all-one function). Let us denote by $\oplus$ ($\wedge$) F_2 -addition (F_2 -multiplication) and by $\text{span } F$ the linear span of the set of Boolean functions F . Let V be a set of vertices, then f_V is $\oplus_{v \in V} f_v$.

Let f be a Boolean function of n variables and F be a set of n -variable Boolean functions. Let the function $\Delta_i f$ be defined in the following way:

$$(\Delta_i f)(x_1, \dots, x_n) = f(0, \dots, 0, 1, x_{i+1}, \dots, x_n) \oplus f(0, \dots, 0, 0, x_{i+1}, \dots, x_n)$$

Let Δf be a set $\{\Delta_i f, 1 \leq i \leq n\}$ and ΔF be the union $\bigcup_{f \in F} \Delta f$. Let $\Delta^j F$ be $\Delta \Delta^{j-1} F$ for $j \geq 1$ and $\Delta^0 F$ be F and $\Delta^* f$ be an union $\bigcup_{i=0}^{\infty} \Delta^i \{f\}$. Note that $\Delta^* f = \bigcup_{i=0}^n \Delta^i \{f\}$ and that $\Delta^* f$ is the smallest set of Boolean functions containing f and closed under all operations Δ_i ($1 \leq i \leq n$).

Let $\diamond_i f$ be the set of all n -variable boolean functions g for which there exists constants $c_1, \dots, c_i$ such that $g(x_1, \dots, x_n) = f(c_1, \dots, c_i, x_{i+1}, \dots, x_n)$. Let $\diamond f$ be equal to $\bigcup_{i=0}^n \diamond_i f$.

3 Size Minimality

Let the permutation π of variables be w.l.o.g. an identity in the whole section.

Theorem 1. *Let B be a $\oplus$ AOBDD ($\oplus$ OBDD, $\oplus$ BDD). Then there exists a $\oplus$ AOBDD ($\oplus$ OBDD, $\oplus$ BDD) representing the same function and of the same size without negative arcs.*

The proof proceeds as follows: We replace each negative arc by a pair of a positive arc and an unlabelled arc. Then the unlabelled arcs are transformed so that they only start in the source. The details are omitted due to space limitation. In the rest of the section we consider only $\oplus$ AOBDDs without negative arcs:

Lemma 1. *Let V be a set of vertices not containing the source. Then for each $1 \leq i \leq n$ there exists a set W not containing the source such that $\Delta_i f_V = f_W$.*

Lemma 2. *Let f be an n -variable Boolean function. There are uniquely determined n -variable Boolean functions $h_1, \dots, h_n$ and constant c which satisfy:*

- The function h_i does not essentially depend on the first i variables.
- $f = c \oplus \bigoplus_{i=1}^n (x_i \wedge h_i)$

The function h_i is equal to $\Delta_i f$ and the constant c is equal to $f(0, \dots, 0)$.

Theorem 2. *Let B be the minimal shared $\oplus$ AOBDD representing $f_1, \dots, f_k$. Then the number of its vertices is equal to $k + \dim_{F_2} \text{span } \bigcup_{i=1}^k \Delta^* f_i$*

Proof. We give only a sketch of the proof for a non-shared $\oplus$ AOBDD representing f . Let V be the vertices where an odd number of arcs from the source lead to. Because all arcs from the source are unlabelled it is clear that $f_V = f_B$. From Lemma 1 it follows that for each function $g \in \Delta^* f$ there exists a set of vertices W not containing the source such that $g = f_W$; thus the set W exists also for every $g \in \text{span } \Delta^* f$. Thus there must be at least $\dim_{F_2} \text{span } \Delta^* f$ vertices distinct from the source and at least $1 + \dim_{F_2} \text{span } \Delta^* f$ vertices altogether.

Now, it remains to prove that this number of vertices is sufficient. If the function is all-zero, the theorem is trivial; otherwise $I \in \Delta^* f$ where I is the all-one Boolean function. Let $\Delta^{*(i)} f$ be those functions which belong to $\Delta^* f$ and do not essentially depend on the variables $x_1, \dots, x_i$ (remember π is an identity), clearly $\Delta^* f = \Delta^{*(0)} f \supseteq \Delta^{*(1)} f \supseteq \dots \supseteq \Delta^{*(n)} f \supseteq \{I\}$. Let $f_1, \dots, f_k$ be a basis of $\text{span } \Delta^* f$ with the following property: For each i there exists j that $f_j, \dots, f_k$ is a basis of $\text{span } \Delta^{*(i)} f$. Clearly f_k is I and w.l.o.g. $f_i(0, \dots, 0) = 0$ for $i < k$. One can construct a $\oplus$ AOBDD which contains only vertices representing $f_1, \dots, f_k$ and the source; the construction is omitted due to space limitation.

The functions in $\text{span } \Delta^* f$ need not to be expressible as the linear combinations of represented functions. The previous theorem gives only the formula for the size of the minimal diagram. The next theorem was proved in [8]:

Theorem 3. *Let B be the size-minimal shared $\oplus$ OBDD representing functions $f_1, \dots, f_k$. Then its size is equal to $k + \dim_{F_2} \text{span } \cup_{i=1}^k \diamond f_i$.*

Since $\text{span } \triangle^* f \subseteq \text{span } \diamond f$, the size of the size-minimal $\oplus$ AOBDD representing f is at most the size of the size-minimal $\oplus$ OBDD. The relation between the sizes of $\oplus$ AOBDD and $\oplus$ OBDD is described in the next theorem. All the $\oplus(A)$ OBDDs considered since now may contain both positive and negative arcs.

Lemma 3. *Let $h_n(x_1, \dots, x_n) = x_1 \oplus \dots \oplus x_n$. Every $\oplus$ BDD, in particular every $\oplus$ OBDD, representing function h_n has at least $2n$ arcs.*

Theorem 4. *Let f be an arbitrary Boolean function.*

1. *Let B_1 be an arbitrary $\oplus$ OBDD representing the function f . Then there exists a $\oplus$ AOBDD B_2 with at most the same number of vertices as B_1 has and with at most the same number of arcs as B_1 .*
2. *Let B_1 be an arbitrary $\oplus$ AOBDD representing the function f . Then there exists a $\oplus$ OBDD B_2 with at most $O(n)$ -times more vertices than B_1 has and with at most $O(1)$ -times more arcs than B_1 .*

All bounds given in this theorem are asymptotically sharp.

4 Parity AOBDD and Parity OBDD Uniqueness Properties

In order to formulate the uniqueness conditions let us define the *PDFS algorithm* (P states for priority): The algorithm is the usual graph depth-first search algorithm started from the source with one additional rule:

- In case of $\oplus$ AOBDDs: if there are more possibilities to select an arc to continue through it always continues through the arc with the π -greatest variable and it prefers an arc leading to the sink among all such arcs.
- In case of $\oplus$ OBDDs: if there are more possibilities to select an arc to continue through it always continues through the arc leading to the vertex labelled with the π -greatest variable; if there is an arc leading to the sink it prefers this arc to any other.

We call the (rooted) tree with labelled arcs produced by the algorithm *PDFS-tree*. There might exist more different PDFS-trees for one $\oplus(A)$ OBDD. The ordering of the vertices *induced by PDFS* is the ordering induced by the pre-order listing of the vertices in the PDFS-tree. We call a $\oplus(A)$ OBDD *linearly reduced* if functions represented by its vertices different from the source are linearly independent. We say that a parity (A)OBDD satisfies *the uniqueness conditions* if it satisfies the following conditions:

- It is linearly reduced.
- It contains no negative arcs.

- Its PDFS-tree is unique.
- In case of $\oplus$ AOBDDs: It contains at most one unlabelled arc to a non-sink vertex and at most one to the sink; in particular the degree of its source is at most two.
- In case of $\oplus$ OBDDs: If there is a tree arc leading from its vertex v to the vertex labelled with x_i then there are no other arcs leading from v to any vertex labelled with x_i .

Notice that in case of $\oplus$ OBDDs there can be either exactly one tree arc leading from v to a vertex labelled with x_i (and no other arcs) or any number of non-tree arcs leading from v to vertices labelled with x_i .

Theorem 5. *For each Boolean function f and any π there exists a π - $\oplus$ AOBDD which satisfies the uniqueness conditions.*

Theorem 6. *Any π - $\oplus$ AOBDD B which satisfies the uniqueness conditions is size-minimal.*

Theorem 7. *Let B_1 and B_2 be two π - $\oplus$ AOBDDs which satisfy the uniqueness conditions and represent the same function f . Let $v_0^1, v_1^1, \dots, v_m^1$ be the vertices of B_1 and $v_0^2, v_1^2, \dots, v_m^2$ be the vertices of B_2 in the ordering induced by PDFS¹. Then their PDFS-trees are isomorphic and it holds for each $1 \leq k \leq m$:*

$$\text{span} \{f_{v_i^1}, 1 \leq i \leq k\} = \text{span} \{f_{v_i^2}, 1 \leq i \leq k\}$$

The $\oplus$ AOBDDs satisfying the uniqueness conditions representing the same boolean function have in some sense the same structure. Unfortunately the equality of the linear spans does not imply that $f_{v_i^1} = f_{v_i^2}$ for all $1 \leq i \leq m$; one can find non-isomorphic parity AOBDDs representing the same function which both satisfy the uniqueness conditions. The proofs of the next theorems π - $\oplus$ OBDDs satisfying the uniqueness conditions are omitted due to space limitation.

Theorem 8. *For each Boolean function f and any π there exists a π - $\oplus$ OBDD representing f which satisfies the uniqueness conditions.*

Theorem 9. *Each π - $\oplus$ OBDD B which satisfies the uniqueness conditions is size-minimal.*

Theorem 10. *Let B_1 and B_2 two π - $\oplus$ OBDDs which satisfy the uniqueness conditions and represent the same function f . Then their PDFS-trees and the diagrams themselves are isomorphic. Moreover, let $v_0^1, \dots, v_m^1$ be the sequence of B_1 's vertices induced by PDFS and $v_0^2, \dots, v_m^2$ be the sequence of B_2 's vertices induced by PDFS, then $f_{v_i^1} = f_{v_i^2}$ for all $0 \leq i \leq m$.*

¹ The numbers of the vertices are equal due to Theorem 6.

5 Data Structures Representing Parity (A)OBDDs

A parity OBDD of size² S without negative arcs can be represented by a $S \times S$ boolean matrix whose rows and columns are indexed by vertices of the $\oplus$ OBDD; its entry is one iff there is an arc leading from the row-vertex to the column-vertex. In case of parity AOBDD the matrix columns are indexed by pairs of vertices of the $\oplus$ AOBDD and variables. The matrix representation can be adapted to case of $\oplus$ (A)OBDDs with both positive and negative arcs.

Table 1. Running times for functions supported by used data structures, m is the size of the represented matrix

Function	Standard representation	Lazy representation
element	$O(1)$	$O(m/\log m)$
newspecial	$O(m)$	$O(m^2/\log m)$
add	$O(m)$	$O(m/\log m)$
exchange	$O(m^2)$	$O(m)$
Used space	$O(m^2)$	$O(m^3/\log m)$

We will use two data structures for matrix representation of $\oplus$ (A)OBDDs — we call them *standard representation* and *lazy representation*; they differ at speed and storage size (see Table 1). Each of them supports the following functions:

- **element**(i, j) This function returns the entry at the i -th row and the j -th column.
- **newspecial**(v) This function declares v as a special vector; the size of v should be equal to the row-size. The number of special vectors must be bounded by a linear function of the number of matrix rows.
- **add**(i, j) This function adds the j -th special vector to the i -th row.
- **exchange**(i, j) This function exchange the i -th and the j -th row in the matrix.

Both the data structures represent a matrix and allow to add some of vectors, those declared as special ones, to matrix rows. The standard representation maintain a matrix of the appropriate size and the set of special vectors; all its functions are implemented straightforwardly. The lazy representation uses ideas similar to those used in [1]. For the overview of running times see Table 1. The details of both data structures are omitted due to space limitation.

6 Algorithms for Parity (A)OBDDs

In this section we discuss basic algorithms for parity (A)OBDDs. They include: *Evaluation* — evaluation of the represented function for the given assignment

² Recall that S is the size of $\oplus$ (A)OBDD, i.e. the number of its vertices.

of Boolean values to the variables, *Removal of negative-arcs* — modification of a $\oplus(A)$ OBDD in order to get rid of all negative arcs (without enlarging its size), *Linear reduction* — modification of a $\oplus(A)$ OBDD to a linearly reduced one, *Unification* — modification of a $\oplus(A)$ OBDD to one which satisfies the uniqueness conditions and *PDFS algorithm* itself. The *Minimization* algorithm can be implemented by one call of Unification since the diagram which satisfies the uniqueness conditions is size-minimal. The achieved running times and used space of the algorithms are presented in Table 2 and differs according to used data structure. All the algorithms are easily adaptable to case of shared $\oplus(A)$ OBDDs yielding the same running time. In the rest of the section, let S be the size, T the number of arcs and n the number of variables of an input parity (A)OBDD.

Table 2. Running times for presented algorithms (S is the number of vertices, T is the number of arcs and n is the number of variable of a parity (A)OBDD)

Algorithm	Parity OBDDs		Parity AOBDDs	
Evaluation	$O(T)$		$O(T)$	
The PDFS algorithm	$O(T + n)$		$O(T + n)$	
Used representation	Standard	Lazy	Standard	Lazy
Space complexity	$O(S^2)$	$O(S^3/\log S)$	$O(S^2)$	$O(S^3/\log S)$
Removal of negative-arcs	$O(S^3)$	$O(S^3/\log S)$	$O(S^3)$	$O(S^3/\log S)$
Linear reduction	$O(S^3)$	$O(S^3/\log S)$	$O(S^3)$	$O(S^3/\log S)$
Unification	$O(S^3)$	$O(S^3/\log S)$	$O(S^3)$	$O(S^3/\log S)$
Minimization	$O(S^3)$	$O(S^3/\log S)$	$O(S^3)$	$O(S^3/\log S)$

6.1 Evaluation and the PDFS Algorithm

Implementation of Evaluation using DFS approach in time $O(T)$ is trivial (see also [5], [8]). The implementation of PDFS algorithm running in time $O(T + n)$ uses the bucket-sort algorithm; details are omitted due to space limitation.

6.2 Operation Reexpress

An essential operation is the operation *reexpress*(w, W) where w is a vertex (different from the source and the sink) of a $\oplus(A)$ OBDD and W is a set of its vertices containing the source, the sink or w . We omit some details due to space limitation. Let the *rank* of a vertex of a $\oplus$ OBDD ($\oplus$ AOBDD) is the variable which it is labelled with (the π -smallest variable labelled to any of arcs leading from it). We demand in case of $\oplus$ OBDDs ($\oplus$ AOBDDs) that the rank of all vertices in W is the same (equal or π -greater) as the rank of w . The goal of *reexpress* is to change f_w to $\bigoplus_{u \in W} f_u$ and modify the $\oplus(A)$ OBDD in order not to affect functions represented by other vertices. If a $\oplus(A)$ OBDD is linearly reduced, it is also linearly reduced after performing the operation. The implementation is following: Add (in the sense of F_2 addition) all rows

corresponding to vertices of W different from w to the row representing w . Add (in the sense of F_2 addition) the column corresponding to w to the columns corresponding to $u \in W \setminus \{w\}$, in case of $\oplus$ AOBDDs in each of n parts separately. This can be done by adding a certain row, which we declare as a special one in the used data structure, to some of the matrix rows.

If at most $O(S)$ operations **reexpress** are performed, then the operation **reexpress** requires time $O(S^2)$ using standard representation or $O(S^2/\log S)$ using lazy representation for $\oplus$ OBDDs and $O(nS^2)$ using standard representation or $O(nS^2/\log S)$ using lazy representation for $\oplus$ AOBDDs.

6.3 Removal of Negative Arcs, Linear Reduction

The algorithm for removal of negative arcs proceeds in the same manner as the proof of Theorem 1 but we leave out the details due to space limitation. The algorithm for linear reduction follows the ideas of Waack's algorithm of $\oplus$ OBDDs (see [8]) but we utilize the possibility to remove negative arcs.

We get rid of negative arcs and sort the vertices according to their rank (using the bucket-sort algorithm) into a π -decreasing sequence — $v_1, \dots, v_n$; we find the first i such that $f_{v_1}, \dots, f_{v_i}$ are not linearly independent and express f_{v_i} as a linear combination of $f_{v_1}, \dots, f_{v_{i-1}}$: $f_{v_i} = \bigoplus_{w \in W} f_w$. Then we call **reexpress**($v_i, W \cup \{v_i\}$) (the call is legal due to Lemma 4) — the new function represented by v_i is the all-zero function and thus v_i can be removed. To check the linear independence we use a modification of Gaussian elimination procedure Lemma 5 (its proof is in [8]) and Lemma 6 are essential in the whole process:

Lemma 4. *Consider a $\oplus$ OBDD which does not contain negative arcs and let $f_v = \bigoplus_{w \in W} f_w$, let $f_w, w \in W$, be linearly independent functions and let the rank of v be the π -smallest among the ranks of vertices in W . Then the rank of all vertices in W is equal to the rank of v .*

Lemma 5. *Let W be a set of vertices of a $\oplus$ OBDD of the same rank and let all the functions represented by vertices with the rank π -greater than the rank of vertices in W be linearly independent. Functions $f_w, w \in W$ are linearly independent iff their rows in the matrix representation are linearly independent.*

Lemma 6. *Let W be a set of vertices of a $\oplus$ AOBDD, let w_0 be the vertex in W with the π -smallest rank and let all functions represented by vertices with the π -greater rank than w_0 be linearly independent. Functions $f_w, w \in W$ are linearly independent iff their rows in the matrix representation are linearly independent.*

6.4 Unification

The idea of Unification is simple — run the PDFS algorithm and change the structure of the $\oplus$ (A)OBDD if the uniqueness conditions are violated. We call

Removal of zero arcs and Linear reduction first, then we start the PDFS algorithm and continue running it until we encounter the first violation of the uniqueness conditions, that means we want to continue the PDFS algorithm from a vertex and there are:

- two or more unvisited vertices of the same rank and there are no unvisited vertices of π -greater rank, in case of $\oplus$ OBDDs.
- two or more arcs labelled with the same variable (or unlabelled in case that we are in the source) leading to unvisited vertices different from the sink and there are no arcs labelled with a π -greater variable leading to unvisited vertices, in case of $\oplus$ AOBDDs.

Let v be the vertex where the uniqueness conditions are violated, i.e. the vertex from which the algorithm PDFS cannot uniquely continue. In case of $\oplus$ OBDDs, let W be the set of (both unvisited and visited) vertices of the same rank violating the uniqueness conditions to which an arc leads from v and let w be any unvisited vertex of W . In case of $\oplus$ AOBDDs, let W be the set of all unvisited vertices violating the uniqueness conditions (note that arcs leading from v to any vertex of W are labelled with the same variable, say y) and let w be any member of W with the π -smallest rank among vertices in W . Call $\text{reexpress}(w, W)$. This ensures there is no more any violation of the uniqueness conditions at the vertex v , since in case of $\oplus$ OBDD w is the only son of v with its rank and in case of $\oplus$ AOBDD w is the only unvisited son (different from the sink) of v to which an arc labelled with y leads. Thus the PDFS algorithm can continue through the arc to the vertex w . Unification ends when we have created the PDFS-tree of the whole $\oplus(A)$ OBDD. Now remove all vertices not accessible from the source, i.e. not included to the PDFS-tree.

Acknowledgement. I am indebted to Petr Savický for his carefull reading of early versions of this paper, his suggestions improving the clarity of the statements and the style of the paper.

References

1. Arlazarov, L., Dinic, E. A., Kronrod, A., Faradzev, I. A.: On economical construction of the transitive closure of a directed graph. Dokl. Akad. Nauk USSR 1970, 194, pp. 487–488 (in Russian), Soviet. Math. Dokl. 11, pp. 1209–1210 (in English)
2. Bryant, R. E.: Graph-based algorithms for Boolean function manipulation. IEEE Trans. on Computers 1986, 35, pp. 677–691
3. Bryant, R. E.: Symbolic Boolean manipulation with ordered binary decision diagrams. ACM Comp. Surveys 1992, 24, pp. 293–318
4. Coppersmith, D., Winograd, S.: Matrix multiplication via arithmetic progressions. J. Symbolic Computation 1990, 9, pp. 251–280
5. Gergov, J., Meinel, Ch.: Mod-2-OBDDs — a data structure that generalizes EXOR-Sum-of-Products and Ordered Binary Decision Diagrams. Formal Methods in System Design 1996, 8, pp. 273–282
6. Král', D.: Algebraic and Uniqueness Properties of Parity Ordered Binary Decision Diagrams and their Generalization. ECCC report TR00-013

7. Löbbling, M., Sieling, D., Wegener, I.: Parity OBDDs cannot be handled efficiently enough. *Information Processing Letters* 1998, 67, pp. 163–168
8. Waack, St.: On the descriptive and algorithmic power of parity ordered binary decision diagrams. *Proc. 14th STACS 1997, Lecture Notes in Computer Sci.* 1200, Springer Verlag 1997, pp. 201–212
9. Wegener, I.: *Branching Programs and Binary Decision Diagrams — Theory and Applications*. To appear (2000) in the SIAM monograph series *Trends in Discrete Mathematics and Applications* (ed. P. L. Hammer).
10. Wegener, I.: Efficient data structures for Boolean functions. *Discrete Mathematics* 1994, 136, pp. 347–372

Formal Series over Algebras

Werner Kuich*

Technische Universität Wien
Wiedner Hauptstraße 8–10, A–1040 Wien
kuich@tuwien.ac.at

Abstract. We define two types of series over Σ -algebras: formal series and, as a special case, term series. By help of term series we define systems (of equations) that have tuples of formal series as solutions. We then introduce finite automata and polynomial systems and show that they are mechanisms of equal power. Morphisms from formal series into power series yield combinatorial results.

1 Introduction and Preliminaries

This paper has connections to some sections of Courcelle [3] and owes much to Courcelle [2,3]; moreover, the examples are taken from Courcelle [3]. We first introduce in the preliminary Section 1 the basic algebraic structure: the distributive Ω -monoids of Kuich [11]. These are a generalization of the distributive F -magmas of Section 10 of Courcelle [2].

In Section 2, we consider formal series over Σ -algebras and consider systems (of equations). These systems have least solutions that are obtained by application of the Fixpoint Theorem. The last result of Section 2 gives a combinatorial application of our theory.

In Section 3, we introduce finite automata and polynomial systems and show that they are mechanisms of equal power. Normal forms for finite automata and polynomial systems are discussed.

A commutative monoid $\langle A, +, 0 \rangle$ is *naturally ordered* iff the set A is partially ordered by the relation $\sqsubseteq$: $a \sqsubseteq b$ iff there exists a c such that $a + c = b$.

A commutative monoid $\langle A, +, 0 \rangle$ is called *complete* iff it is possible to define sums for all families $(a_i \mid i \in I)$ of elements of A , where I is an arbitrary index set, such that the following conditions are satisfied:

- (i) $\sum_{i \in \emptyset} a_i = 0$, $\sum_{i \in \{j\}} a_i = a_j$, $\sum_{i \in \{j,k\}} a_i = a_j + a_k$, for $j \neq k$,
- (ii) $\sum_{j \in J} (\sum_{i \in I_j} a_i) = \sum_{i \in I} a_i$, if $\bigcup_{j \in J} I_j = I$ and $I_j \cap I_{j'} = \emptyset$ for $j \neq j'$.

A complete naturally ordered monoid $\langle A, +, 0 \rangle$ is called *continuous* iff for all index sets I and all families $(a_i \mid i \in I)$ in A the following condition is satisfied (see Goldstern [4], Karner [6], Kuich [10]):

$$\sup \left\{ \sum_{i \in E} a_i \mid E \subseteq I, E \text{ finite} \right\} = \sum_{i \in I} a_i.$$

* Partially supported by Wissenschaftlich-Technisches Abkommen Österreich-Ungarn.

Here and in the sequel, “sup” denotes the least upper bound with respect to the natural order.

We now come to the central notion of this section. Let $\langle A, +, 0 \rangle$ be a commutative monoid. Let $\Omega = (\omega_i \mid i \in I)$ be a family of finitary operations on A indexed by an index set I . Let $\Omega_k = (\omega_i \mid i \in I_k)$ be the family of k -ary operations, $k \geq 0$, indexed by $I_k \subseteq I$.

The algebra $\langle A, +, 0, \Omega \rangle$, where $\langle A, +, 0 \rangle$ is a commutative monoid, is called a *distributive Ω -monoid* iff the following two conditions are satisfied for all $\omega \in \Omega_k$ and all $a, a_1, \dots, a_k \in A$, $k \geq 1$:

- (i) $\omega(a_1, \dots, a_{j-1}, 0, a_{j+1}, \dots, a_k) = 0$ for all $1 \leq j \leq k$,
- (ii) $\omega(a_1, \dots, a_{j-1}, a_j + a, a_{j+1}, \dots, a_k) =$
 $\omega(a_1, \dots, a_{j-1}, a_j, a_{j+1}, \dots, a_k) + \omega(a_1, \dots, a_{j-1}, a, a_{j+1}, \dots, a_k)$
 for all $1 \leq j \leq k$.

A distributive Ω -monoid $\langle A, +, 0, \Omega \rangle$ is briefly denoted by A if $+, 0$ and Ω are understood. Similar algebras are considered in Courcelle [2] and Bozapalidis [1].

A distributive Ω -monoid $\langle A, +, 0, \Omega \rangle$ is termed *naturally ordered* iff $\langle A, +, 0 \rangle$ is naturally ordered.

A distributive Ω -monoid $\langle A, +, 0, \Omega \rangle$ is called *complete* iff $\langle A, +, 0 \rangle$ is complete and the following condition is satisfied for all $\omega \in \Omega_k$, index sets I , $a_1, \dots, a_{j-1}, a_{j+1}, \dots, a_k \in A$, $1 \leq j \leq k$, $k \geq 1$, $b_i \in \omega$, $i \in I$:

$$\omega(a_1, \dots, a_{j-1}, \sum_{i \in I} b_i, a_{j+1}, \dots, a_k) = \sum_{i \in I} \omega(a_1, \dots, a_{j-1}, b_i, a_{j+1}, \dots, a_k).$$

Eventually, a complete distributive Ω -monoid $\langle A, +, 0, \Omega \rangle$ is called *continuous* iff $\langle A, +, 0 \rangle$ is continuous.

Let $\langle A, +, 0 \rangle$ be a naturally ordered commutative monoid. A sequence $(a_i \mid i \in \mathbb{N})$ in A is called ω -chain iff $a_i \sqsubseteq a_{i+1}$, $i \geq 0$. By Theorem 2.3 of Kuich [11], the least upper bounds of ω -chains in A exist if A is a continuous monoid.

Let $\langle A, +, 0 \rangle$ and $\langle A', +, 0 \rangle$ be continuous monoids. A mapping $f : A \rightarrow A'$ is called ω -continuous iff, for each ω -chain $(a_i \mid i \in \mathbb{N})$ in A ,

$$f(\sup(a_i \mid i \in \mathbb{N})) = \sup(f(a_i) \mid i \in \mathbb{N}).$$

2 Distributive Σ -Monoids and Systems of Equations

In this section we introduce Σ -algebras and consider formal series and, as a special case, term series over Σ -algebras. By help of term series we define systems (of equations). The components of their least solutions are formal series.

A *signature* Σ is a set, whose elements are called operation symbols, together with an *arity function*, assigning to each operation symbol its finite *arity*. The signature is uniquely determined by the family $(\Sigma_k \mid k \in \mathbb{N})$, where Σ_k is the set of operation symbols of arity $k \geq 0$. A Σ -algebra is a pair $\langle G, \Sigma^G \rangle$, where G is a nonempty set and $\Sigma^G = (\sigma^G \mid \sigma \in \Sigma)$ is a family of operations such that

$\sigma^G : G^k \rightarrow G$ if $\sigma \in \Sigma_k$ $k \geq 0$. In the sequel, Σ always denotes a signature determined by the family $(\Sigma_k \mid k \in \mathbb{N})$. A Σ -algebra $\langle G, \Sigma^G \rangle$ is often denoted by G .

Let Y be an alphabet of variables, $\Sigma \cap Y = \emptyset$ ($\emptyset$ denotes the empty set). Then the set $T_\Sigma(Y)$ of *terms over Y* is the least set of words over the alphabet $\Sigma \cup Y \cup \{(\cdot, \cdot)\} \cup \{, \}$ such that $Y \subseteq T_\Sigma(Y)$ and for all $\sigma \in \Sigma_k$, $k \geq 0$, and $t_j \in T_\Sigma(Y)$, $1 \leq j \leq k$, the word $\sigma(t_1, \dots, t_k)$ belongs to $T_\Sigma(Y)$. Define, for $\sigma \in \Sigma_k$, $k \geq 0$, the operation $\sigma^T : T_\Sigma(Y)^k \rightarrow T_\Sigma(Y)$ by $\sigma^T(t_1, \dots, t_k) = \sigma(t_1, \dots, t_k)$, $t_j \in T_\Sigma(Y)$, $1 \leq j \leq k$. Then $\langle T_\Sigma(Y), \Sigma^T \rangle$, where $\Sigma^T = (\sigma^T \mid \sigma \in \Sigma)$ is a Σ -algebra, called Σ -termalgebra.

Given a Σ -algebra G , we now define formal series. Let $\langle A, +, \cdot, 0, 1 \rangle$ be a semiring. Mappings $r : G \rightarrow A$ are called *formal series (over G)* and are written as a formal infinite sum $r = \sum_{g \in G} (r, g)g$, where the value of the mapping r with argument g is denoted by (r, g) and is called *coefficient* of g . The collection of all these formal series is denoted by $A\langle\langle G \rangle\rangle$.

We now define operations on formal series. Let $r_1, r_2 \in A\langle\langle G \rangle\rangle$. Then the sum $r_1 + r_2$ is again a formal series in $A\langle\langle G \rangle\rangle$ defined by

$$r_1 + r_2 = \sum_{g \in G} ((r_1, g) + (r_2, g))g.$$

The formal series $0 \in A\langle\langle G \rangle\rangle$ is defined by $0 = \sum_{g \in G} 0 \cdot g$. It is clear that $\langle A\langle\langle G \rangle\rangle, +, 0 \rangle$ is a commutative monoid. We will now define operations such that $A\langle\langle G \rangle\rangle$ becomes a distributive Σ -monoid. Let $\sigma \in \Sigma_k$, $k \geq 0$. Then $\hat{\sigma} : A\langle\langle G \rangle\rangle^k \rightarrow A\langle\langle G \rangle\rangle$ is defined by

$$\hat{\sigma}(r_1, \dots, r_n) = \sum_{\substack{g_j \in G \\ 1 \leq j \leq n}} (r_1, g_1) \dots (r_n, g_n) \sigma^G(g_1, \dots, g_n)$$

for $r_j \in A\langle\langle G \rangle\rangle$, $1 \leq j \leq n$. Let $\hat{\Sigma} = \{\hat{\sigma} \mid \sigma \in \Sigma\}$. Then $\langle A\langle\langle G \rangle\rangle, 0, +, \hat{\Sigma} \rangle$ is a distributive Σ -monoid.

The *support* $\text{supp}(r)$ of a formal series r is defined by $\text{supp}(r) = \{t \mid (r, t) \neq 0\}$. Formal series with finite support are called (*formal*) *polynomials*. The collection of all polynomials of $A\langle\langle G \rangle\rangle$ is denoted by $A\langle G \rangle$ and is also a distributive Σ -monoid. Let $g \in G$. Then g also denotes the formal series with support $\{g\}$ where the coefficient of g is 1.

We now consider the Σ -termalgebra $T_\Sigma(Y)$ over Y . Formal series in $A\langle\langle T_\Sigma(Y) \rangle\rangle$ are called *term series*. Let $y_1, \dots, y_n$ be variables in Y . A term series $r \in A\langle\langle T_\Sigma(\{y_1, \dots, y_n\}) \rangle\rangle$ induces a mapping $\hat{r} : A\langle\langle G \rangle\rangle^n \rightarrow A\langle\langle G \rangle\rangle$ defined as follows:

- (i) $\hat{y}_i(r_1, \dots, r_n) = r_i$, $1 \leq i \leq n$,
- (ii) $\hat{t}(r_1, \dots, r_n) = \hat{\sigma}(\hat{t}_1(r_1, \dots, r_n), \dots, \hat{t}_k(r_1, \dots, r_n))$, $t = \sigma(t_1, \dots, t_k) \in T_\Sigma(Y)$, $t_j \in T_\Sigma(Y)$, $1 \leq j \leq k$, $\sigma \in \Sigma_k$, $k \geq 0$,

and

$$\hat{r}(r_1, \dots, r_n) = \sum_{t \in T_\Sigma(Y)} (r, t) \hat{t}(r_1, \dots, r_n),$$

for $r_j \in A\langle\langle G \rangle\rangle$, $1 \leq j \leq n$. For $n = 0$, $\hat{r}$ is a constant formal series in $A\langle\langle G \rangle\rangle$.

Observe that, for $t \in T_\Sigma(Y)$, $\hat{t}(r_1, \dots, r_n) = h(t)$, where h is the morphism defined uniquely by $h(y_j) = r_j$, $1 \leq j \leq n$. If A is commutative, we call the mapping $\hat{r}$ *substitution* and often denote $\hat{r}(r_1, \dots, r_n)$ by $r[r_1/y_1, \dots, r_n/y_n]$. (But observe that the operation symbols $\sigma \in \Sigma$ of r have to be replaced by the corresponding operations $\hat{\sigma} \in \hat{\Sigma}$.) In the sequel, we often denote the mapping $\hat{r}$ induced by the term series r simply by r . This should not lead to any confusion.

Let now G be a Σ -algebra and A be a continuous semiring. Then $A\langle\langle G \rangle\rangle$ is a continuous distributive Σ -monoid. This is easily shown by the definition of infinite sums:

For $r_i \in A\langle\langle G \rangle\rangle$, $s \in S$, $i \in I$, where I is an arbitrary index set,

$$\sum_{i \in I} r_i = \sum_{g \in G} \left(\sum_{i \in I} (r_i, g) \right) g.$$

In the sequel, A will always denote a *continuous commutative* semiring. Hence, $A\langle\langle G \rangle\rangle$ and $A\langle\langle T_\Sigma(Y) \rangle\rangle$ will always be continuous distributive Σ -monoids.

By Theorem 2.16, Lemma 2.17 and Theorem 2.18 of Kuich [11] substitution is an ω -continuous mapping.

Theorem 2.1 *Let A be a continuous commutative semiring and G be a Σ -algebra. Let $y_1, \dots, y_n$ be variables. Consider a term series $r \in A\langle\langle T_\Sigma(\{y_1, \dots, y_n\}) \rangle\rangle$. Then the mapping*

$$\hat{r} : A\langle\langle G \rangle\rangle^n \rightarrow A\langle\langle G \rangle\rangle$$

induced by r is an ω -continuous mapping.

A *system* (with variables $y_1, \dots, y_n$ on the Σ -algebra G) is a sequence of formal equations

$$y_i = r_i(y_1, \dots, y_n), \quad 1 \leq i \leq n, \quad n \geq 1,$$

where each r_i is in $A\langle\langle T_\Sigma(\{y_1, \dots, y_n\}) \rangle\rangle$. A *solution* to the system $y_i = r_i$, $1 \leq i \leq n$, is given by $(\tau_1, \dots, \tau_n) \in A\langle\langle G \rangle\rangle^n$ such that $\tau_i = \hat{r}_i(\tau_1, \dots, \tau_n)$, $1 \leq i \leq n$. A solution $(\tau_1, \dots, \tau_n)$ is termed *least solution* iff $\tau_i \sqsubseteq \tau'_i$, $1 \leq i \leq n$, for all solutions $(\tau'_1, \dots, \tau'_n)$ of $y_i = r_i$, $1 \leq i \leq n$.

Often it is convenient to write the system $y_i = r_i$, $1 \leq i \leq n$, in matrix notation. Defining the two vectors

$$y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \quad \text{and} \quad r = \begin{pmatrix} r_1 \\ \vdots \\ r_n \end{pmatrix},$$

we can write our system in matrix notation

$$y = r(y) \quad \text{or} \quad y = r.$$

Let now $r \in A\langle\langle T_\Sigma(\{y_1, \dots, y_n\}) \rangle\rangle^n$, $r = (r_1(y_1, \dots, y_n), \dots, r_n(y_1, \dots, y_n))$. Then r induces a mapping $\hat{r} : A\langle\langle G \rangle\rangle^n \rightarrow A\langle\langle G \rangle\rangle^n$ by $\hat{r}(\tau_1, \dots, \tau_n)_i = \hat{r}_i(\tau_1, \dots, \tau_n)$, $\tau_i \in A\langle\langle G \rangle\rangle$, $1 \leq i \leq n$, i.e., the i -th component of the value of $\hat{r}$ at $(\tau_1, \dots, \tau_n)$ is given by the value of the i -th component of $\hat{r}$ at $(\tau_1, \dots, \tau_n)$.

A *solution* to $y = r(y)$ is now given by a vector $\tau \in A\langle\langle G \rangle\rangle^n$ such that $\tau = \hat{r}(\tau)$. A solution τ of $y = r$ is termed *least solution* iff $\tau \sqsubseteq \tau'$ for all solutions τ' of $y = r$. Since the mappings induced by $\hat{r}_i$, $1 \leq i \leq n$, are ω -continuous, the mapping $\hat{r}$ is also ω -continuous. Consider now the system $y = r$. Since the least fixpoint of the mapping $\hat{r}$ is nothing else than the least solution of $y = r$, application of the Fixpoint Theorem (see Wechler [16]) yields the following theorem.

Theorem 2.2 *Let A be a continuous commutative semiring and G be a Σ -algebra. Let $y_1, \dots, y_n$ be variables, and consider the system $y = r$ on G . Then the least solution of $y = r$ exists in $A\langle\langle G \rangle\rangle^n$ and equals the least fixpoint of $\hat{r}$*

$$\text{fix}(\hat{r}) = \sup(\hat{r}^i(0) \mid i \in \mathbb{N}).$$

Theorem 2.2 indicates how we can compute an approximation to the least solution of a system $y = r$.

The *approximation sequence* $(\tau^j \mid j \in \mathbb{N})$, where each $\tau^j \in A\langle\langle G \rangle\rangle^n$, associated to the system $y = r(y)$ is defined as follows:

$$\tau^0 = 0, \quad \tau^{j+1} = \hat{r}(\tau^j), \quad j \in \mathbb{N}.$$

Clearly, $(\tau^j \mid j \in \mathbb{N})$ is an ω -chain and $\text{fix}(\hat{r}) = \sup(\tau^j \mid j \in \mathbb{N})$, i.e., we obtain the least solution of $y = r$ by computing the least upper bound of its approximation sequence.

Observe that our systems are nothing else than a specialization of the linear systems of Kuich [12] (there are now only finitely many variables and no leaf symbols). But the definition of a solution is different: In the linear systems of Kuich [12] the components of a solution are tree series while here the components of a solutions are formal series over G .

In the sequel, $y_1, \dots, y_n$ are variables in Y , and we denote $Y_n = \{y_1, \dots, y_n\}$, $n \geq 1$, $Y_0 = \emptyset$.

A system $y_i = r_i$, $1 \leq i \leq n$, is termed *proper* iff $(r_i, y_j) = 0$ for all $1 \leq i, j \leq n$, i.e., iff the term series r_i have no linear part.

By performing a procedure analogous to that described in Theorem 5.1 of Kuich [10], Proposition 20 of Bozapalidis [1] or Theorem 3.2 of Kuich [11], we get the next result.

Theorem 2.3 *For each system there exists a proper one with the same solution.*

Let G and G' be Σ -algebras and consider a morphism h from G into G' . Such a morphism can be extended to a complete morphism from $A\langle\langle G \rangle\rangle$ into $A\langle\langle G' \rangle\rangle$ by $h(\sum_{g \in G}(r, g)g) = \sum_{g \in G}(r, g)h(g)$. In the next theorem we apply such a morphism h to the least solution τ of a system on G and get the least solution $h(\tau)$ of this system on G' . This theorem is analogous to a result of Mezei, Wright [14] (see also Courcelle [3], Proposition 3.7).

Theorem 2.4 *Let G and G' be Σ -algebras and h be a morphism from G into G' . Consider a system $y_i = r_i$, $1 \leq i \leq n$, on G , $r_i \in A\langle\langle T_\Sigma(Y_n) \rangle\rangle$, with least solution $(\tau_1, \dots, \tau_n)$. Then the system $y_i = r_i$, $1 \leq i \leq n$, considered as a system on G' , has the least solution $(h(\tau_1), \dots, h(\tau_n))$.*

Theorem 2.4 admits a corollary that has applications in combinatorics. (Compare it with Theorem 2 of Kuich [8].) A formal series $r \in \mathbb{N}^\infty\langle\langle G \rangle\rangle$ is called *unambiguous* iff, for all $g \in \text{supp}(r)$, $(r, g) = 1$. A system $y_i = r_i$, $1 \leq i \leq n$, is called *unambiguous* iff its least solution $(\tau_1, \dots, \tau_n)$ has unambiguous components τ_i , $1 \leq i \leq n$.

Corollary 2.5 *Let G and $F = z^m z^*$, for some $m \geq 0$, be Σ -algebras, and φ be a morphism from G into F . Define the mapping $h : \mathbb{N}^\infty\langle\langle G \rangle\rangle \rightarrow \mathbb{N}^\infty\langle\langle z^m z^* \rangle\rangle$, by $h(r) = \sum_{g \in G} (r, g) \varphi(g)$, $r \in \mathbb{N}^\infty\langle\langle G \rangle\rangle$. For $r \in \mathbb{N}^\infty\langle\langle G \rangle\rangle$, $n \geq 0$, define $v_r(n) = \sum_{\varphi(g)=z^n} (r, g)$, and $u_r(n) = \sum_{g \in \text{supp}(r), \varphi(g)=z^n} 1$.*

Let $y_i = r_i$, $1 \leq i \leq n$ be a system on G with least solution $(\tau_1, \dots, \tau_n)$ and let $(f_1(z), \dots, f_n(z))$ be the least solution of the system $y_i = r_i$, $1 \leq i \leq n$, considered as a system on F .

Then

$$f_i(z) = \sum_{n \geq m} v_{\tau_i}(n) z^n, \quad 1 \leq i \leq n.$$

Moreover, if $y_i = r_i$, $1 \leq i \leq n$, is an unambiguous system, then

$$f_i(z) = \sum_{n \geq m} u_{\tau_i}(n) z^n, \quad 1 \leq i \leq n.$$

Example 1 (compare with Examples 2.3 and 3.2 of Courcelle [3]). A *tree* is a connected linear graph (without multiple edges, without loops) having no cycles. A *rooted tree* is a tree in which a node is distinguished and is called *root*. A *planted tree* is a rooted tree in which the root has valency one. A *plane tree* is a rooted tree which is embedded in the plane. (For exact definitions see Harary, Prins, Tutte [5] or Klarner [7]; see also Kuich [9] and Stanley [15].) The set of plane trees is denoted by $\mathbb{P}$. Any two isomorphic plane trees are considered equal. Our signature is $\Sigma = \{||, \text{ext}, 1\}$, where $\Sigma_2 = \{||\}$, $\Sigma_1 = \{\text{ext}\}$, $\Sigma_0 = \{1\}$.

We now define the Σ -algebra $\langle\mathbb{P}; ||, \text{ext}, 1\rangle$ as follows:

- (i) For T_1 and T_2 in $\mathbb{P}$ we let $||(T_1, T_2)$ be the plane tree obtained by fusing the roots of T_1 and T_2 .
- (ii) For T in $\mathbb{P}$ we let $\text{ext}(T)$ be the planted plane tree obtained from T by the addition of a new node that becomes the root of $\text{ext}(T)$, linked by a new edge to the root of T .
- (iii) We denote by 1 the plane tree consisting of a single node, the root.

Let $\mathbb{N}^\infty$ be our basic semiring. We consider *mod r -valent planted plane trees*, $r \geq 2$. That are trees whose nodes assume only valences that are elements of $\{i \mid i \equiv 1 \pmod r\}$. Define, for $i \geq 0$, the term $[y]^i$ over the variable y by

$$[y]^0 = 1, \quad [y]^1 = y, \quad [y]^{i+1} = ||([y]^i, y),$$

consider the polynomial system

$$\begin{aligned} y_1 &= \text{ext}(y_2) + \text{ext}(1), \\ y_2 &= [y_1]^r + |(y_2, [y_1]^r), \end{aligned}$$

on $\mathbb{P}$ and denote its least solution by (τ_1, τ_2) . It can easily be verified that τ_1 is the characteristic series of mod r -valent planted plane trees and τ_2 is the characteristic series of the plane trees whose root has valency congruent 0 modulo r , while all other nodes have a valency congruent 1 modulo r . Hence, by Corollary 2.5, $\tau = \sum_{n \geq 1} u(n)z^n$, where $u(n)$ is the number of mod r -valent planted plane trees with n nodes, is the first component of the least solution of the system

$$\begin{aligned} y_1 &= zy_2 + z^2, \\ y_2 &= \frac{1}{z^{r-1}}y_1^r + \frac{1}{z^r}y_1^r y_2 \end{aligned}$$

on zz^* . Elimination of y_2 yields the equation

$$z^{r+2} - z^r y_1 + y_1^{r+1} = 0.$$

This equation is solved in Kuich [9]. □

3 Finite Automata and Polynomial Systems

In this section, we define finite automata and polynomial systems, and show that the collection of behaviors of finite automata coincides with the collection of components of least solutions of polynomial systems. Moreover, normal forms for finite automata and polynomial systems are defined. The last result of this section is a Kleene Theorem for formal series.

A finite automaton (on G)

$$\mathfrak{A} = (Q, M, S, P)$$

is given by

- (i) a nonempty finite set Q of *states*;
- (ii) a family $M = (M_k \mid 1 \leq k \leq m)$, $m \geq 0$, of *transition matrices* M_k of dimension $Q \times Q^k$, $1 \leq k \leq m$, such that $(M_k)_{q, (q_1, \dots, q_k)} \in A\langle T_\Sigma(\{z_1, \dots, z_k\}) \rangle$ for $q, q_1, \dots, q_k \in Q$; here the z_k , $1 \leq k \leq m$, are variables;
- (iii) an *initial state vector* S of dimension $1 \times Q$ with $S_q \in A\langle T_\Sigma(\{z_1\}) \rangle$, $q \in Q$;
- (iv) a *final state vector* P of dimension $Q \times 1$ with $P_q \in A\langle T_\Sigma(\emptyset) \rangle$, $q \in Q$.

The *approximation sequence* $(\tau^j \mid j \in \mathbb{N})$, associated to $\mathfrak{A}$, where τ^j , $j \geq 0$ is a column vector of dimension $Q \times 1$ with $\tau_q^j \in A\langle\langle G \rangle\rangle$, $q \in Q$, is defined as follows: For $j \geq 0$, $q \in Q$,

$$\begin{aligned} \tau_q^0 &= 0, \\ \tau_q^{j+1} &= \sum_{1 \leq k \leq m} \sum_{q_1, \dots, q_k \in Q} (M_k)_{q, (q_1, \dots, q_k)} [\tau_{q_1}^j / z_1, \dots, \tau_{q_k}^j / z_k] + \hat{P}_q. \end{aligned}$$

Since the $(M_k)_{q, (q_1, \dots, q_k)}$, $1 \leq k \leq m$, $q, q_1, \dots, q_k \in Q$, are ω -continuous mappings the least upper bound τ of the approximation sequence exists. The *behavior* $\|\mathfrak{A}\|$ of the automaton $\mathfrak{A}$ is now defined by $\|\mathfrak{A}\| = S(\tau) \in A\langle\langle G \rangle\rangle$.

Observe that our finite automata are nothing else than a specialization of the tree automata of Kuich [12] (there are now only finitely many states and no leaf symbols). But the definition of the behavior is different: In the automata of Kuich [12] the behavior is a tree series while here the behavior is a formal series over G .

A system $y_i = r_i$, $1 \leq i \leq n$, is termed *polynomial system* iff $r_i \in A(T_\Sigma(Y_n))$ is a polynomial. The collection of the components of least solutions of polynomial systems on G is denoted by $\mathfrak{Equ}(G)$. Observe that $\mathfrak{Equ}(G)$ also depends on the semiring A .

We now define a normal form for polynomial systems. A Σ -term in $T_\Sigma(Y)$ is called *uniform* iff it is of the form $\sigma(y_{i_1}, \dots, y_{i_k})$, $\sigma \in \Sigma$, $k \geq 0$. A polynomial r in $A(T_\Sigma(Y_n))$ is called *uniform* iff each Σ -term in $\text{supp}(r)$ is uniform. A polynomial system $y_i = r_i$, $1 \leq i \leq n$, is said to be in *normal form* iff each r_i is uniform.

A finite automaton $\mathfrak{A} = (Q, (M_k \mid 1 \leq k \leq m), S, P)$ is in *normal form* iff, for all $q, q_1, \dots, q_k \in Q$, $1 \leq k \leq m$, $\text{supp}((M_k)_{q, (q_1, \dots, q_k)}) \subseteq \{\sigma(z_1, \dots, z_k) \mid \sigma \in \Sigma_k\}$.

Theorem 3.1 *For a formal series r in $A\langle\langle G \rangle\rangle$ the following four statements are equivalent:*

- (i) r is in $\mathfrak{Equ}(G)$;
- (ii) r is a component of the least solution of a polynomial system on G in normal form;
- (iii) r is the behavior of a finite automaton on G ;
- (iv) r is the behavior of a finite automaton on G in normal form.

Example 2 (compare with Examples 2.1 and 3.1 of Courcelle [3]). Let $Z = \{z_1, \dots, z_m\}$ be an alphabet. Let $\Sigma = \{\bullet, \varepsilon\} \cup Z$, $\Sigma_2 = \{\bullet\}$, $\Sigma_0 = \{\varepsilon\} \cup Z$. Then $\langle X^*; \cdot, x_1, \dots, x_m, \varepsilon \rangle$, where $X = \{x_1, \dots, x_m\}$, $\cdot$ is concatenation and ε is the empty word, is a Σ -algebra. Consider now the mappings induced by terms t , $t \in T_\Sigma(Y_n)$. Then, for each such term t there exists a word $\alpha \in (X \cup Y_n)^*$ such that the term t and the power series α induce the same mapping $\hat{t}$. (E. g., if $t = \bullet(\bullet(y_1, z_1), \bullet(z_2, y_2))$, $z_1, z_2 \in Z$, then we choose $\alpha = y_1 x_1 x_2 y_2$ and $\hat{t}(r_1, r_2) = \alpha[r_1/y_1, r_2/y_2]$ for $r_1, r_2 \in A\langle\langle X^* \rangle\rangle$.) Hence, for each term series in $A\langle\langle T_\Sigma(Y_n) \rangle\rangle$ there exists a power series in $A\langle\langle (X \cup Y_n)^* \rangle\rangle$ that induces the same mapping acting on $A\langle\langle X^* \rangle\rangle$. This implies that for each polynomial system on X^* there exists an algebraic system (in the sense of Kuich [10], page 623) with the same least solution. This means that $\mathfrak{Equ}(X^*)$ coincides with $A^{\text{alg}}\langle\langle X^* \rangle\rangle$.

Consider now the monoid $\langle z^*, \cdot, z, \dots, z, 1 \rangle$, z a symbol; it is a Σ -algebra. Let $\varphi : X^* \rightarrow z^*$ be defined by $\varphi(w) = z^{|w|}$, $w \in X^*$. Then φ is a morphism and the mapping $h : A\langle\langle X^* \rangle\rangle \rightarrow A\langle\langle z^* \rangle\rangle$, defined by $h(r) = \sum_{w \in X^*} (r, w) z^{|w|}$, $r \in A\langle\langle X^* \rangle\rangle$, is a complete semiring morphism.

Consider now an unambiguous algebraic system $y_i = r_i$, $1 \leq i \leq n$, $r_i \in \mathbb{N}^\infty\langle\langle (X \cup Y_n)^* \rangle\rangle$ with least solution $(\tau_1, \dots, \tau_n)$. Let $(f_1(z), \dots, f_n(z))$ be the

least solution of the algebraic system $y_i = h(r_i)$, $1 \leq i \leq n$, $h(r_i) \in \mathbb{N}^\infty \langle\langle \{z\} \cup Y_n \rangle\rangle$. Then, by Corollary 2.5, $f_i(z)$, $1 \leq i \leq n$, is the structure generating function of the context-free language $\text{supp}(\tau_i)$. Essentially, this is Theorem 2 of Kuich [8]. $\square$

References

1. Bozapalidis, S.: Equational elements in additive algebras. *Theory Comput. Systems* 32(1999) 1–33.
2. Courcelle, B.: Equivalences and transformations of regular systems—Applications to recursive program schemes and grammars. *Theor. Comp. Sci.* 42(1986) 1–122.
3. Courcelle, B.: Basic notions of universal algebra for language theory and graph grammars. *Theoretical Computer Science* 163(1996) 1–54.
4. Goldstern, M.: Vervollständigung von Halbringen. Diplomarbeit, Technische Universität Wien, 1985.
5. Harary F., Prins G., Tutte W. T.: The number of plane trees. *Indagationes Mathematicae* 26(1964) 319–329.
6. Karner, G.: On limits in complete semirings. *Semigroup Forum* 45(1992) 148–165.
7. Klarner D. A.: Correspondences between plane trees and binary sequences. *J. Comb. Theory* 9(1970) 401–411.
8. Kuich, W.: Über die Entropie kontext-freier Sprachen. Habilitationsschrift, Technische Hochschule Wien, 1970.
English translation: On the entropy of context-free languages. *Inf. Control* 16(1970) 173–200.
9. Kuich, W.: Languages and the enumeration of planted plane trees. *Indagationes Mathematicae* 32(1970) 268–280.
10. Kuich, W.: Semirings and formal power series: Their relevance to formal languages and automata theory. In: *Handbook of Formal Languages* (Eds.: G. Rozenberg and A. Salomaa), Springer, 1997, Vol. 1, Chapter 9, 609–677.
11. Kuich, W.: Formal power series over trees. In: *Proceedings of the 3rd International Conference Developments in Language Theory* (S. Bozapalidis, ed.), Aristotle University of Thessaloniki, 1998, pp. 61–101.
12. Kuich, W.: Pushdown tree automata, algebraic tree systems, and algebraic tree series. *Information and Computation*, to appear.
13. Kuich, W., Salomaa, A.: *Semirings, Automata, Languages*. EATCS Monographs on Theoretical Computer Science, Vol. 5. Springer, 1986.
14. Mezei, J., Wright, J. B.: Algebraic automata and context-free sets. *Inf. Control* 11(1967) 3–29.
15. Stanley, R. P.: *Enumerative Combinatorics*, Volume 2. Cambridge University Press, 1999.
16. Wechler, W.: *Universal Algebra for Computer Scientists*. EATCS Monographs on Computer Science, Vol. 25. Springer, 1992.

μ -Calculus Synthesis

Orna Kupferman¹ and Moshe Y. Vardi^{2*}

¹ Hebrew University, The Institute of Computer Science, Jerusalem 91904, Israel
Email: orna@cs.huji.ac.il, URL: <http://www.cs.huji.ac.il/~orna>

² Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email: vardi@cs.rice.edu, URL: <http://www.cs.rice.edu/~vardi>

Abstract. In system synthesis, we transform a specification into a system that is guaranteed to satisfy the specification. When the system is open, it interacts with an environment via input and output signals and its behavior depends on this interaction. An open system should satisfy its specification in all possible environments. In addition to the input signals that the system can read, an environment can also have internal signals that the system cannot read. In the above setting, of *synthesis with incomplete information*, we should transform a specification that refers to both readable and unreadable signals into a system whose behavior depends only on the readable signals. In this work we solve the problem of synthesis with incomplete information for specifications in μ -calculus. Since many properties of systems are naturally specified by means of fixed points, the μ -calculus is an expressive and important specification language. Our results and technique generalize and simplify previous work on synthesis. In particular, we prove that the problem of μ -calculus synthesis with incomplete information is EXPTIME-complete. Thus, it is not harder than the satisfiability or the synthesis problems for this logic.

1 Introduction

In computer system design, we distinguish between *closed* and *open* systems. A *closed system* is a system whose behavior is completely determined by the state of the system. An *open system* is a system that interacts with its environment and whose behavior depends on this interaction. In *system synthesis*, we transform a specification into a system that is guaranteed to satisfy the specification. Earlier work on synthesis considers closed systems. There, a system that meets the specification can be extracted from a constructive proof that the specification is satisfiable [19,5]. As argued in [20], such synthesis paradigms are not of much interest when applied to open systems. For open systems, we should distinguish between output signals (generated by the synthesized system), over which we have control, and input signals (generated by the environment), over which we have no control. While satisfaction of the specification only guarantees that we can synthesize a system that satisfies the specification for some environment (that is, for some behavior of the input signals), we would like to synthesize a system that satisfies the specification for all environments.

* Supported in part by NSF grant CCR-9700061, and by a grant from the Intel Corporation.

We now make this intuition more formal. We first consider the linear approach to synthesis, where the specification describes the set of correct computations of the system. Given sets I and O of input and output signals, respectively, we can view a system as a *strategy* $P : (2^I)^* \rightarrow 2^O$ that maps a finite sequence of sets of input signals into a set of output signals. When P interacts with an environment that generates infinite input sequences, it associates with each input sequence an infinite computation over $2^{I \cup O}$. Given a linear specification ψ over $I \cup O$, *realizability* of ψ is the problem of determining whether there exists a system P all of whose computations satisfy ψ . Synthesis of ψ then amounts to constructing such P [20].

Though the system P is deterministic, it induces a computation tree. The branches of the tree correspond to external nondeterminism, caused by different possible inputs. Thus, the tree has a fixed branching degree $|2^I|$, and it embodies all the possible inputs (and hence also computations) of P^1 . When we synthesize P from an LTL specification ψ , we require ψ to hold in all the paths of P 's computation tree. Consequently, we cannot impose possibility requirements on P (cf. [3]). For example, while we can require that for every infinite sequence of input, the output signal v is eventually assigned true, we cannot require that every finite sequence of inputs can be extended so that v is eventually assigned true. In order to express possibility properties, we should specify P using *branching temporal logics*, which enable both universal and existential path quantification [9]. Given a branching specification ψ over $I \cup O$, *realizability* of ψ is the problem of determining whether there exists a system P whose computation tree satisfies ψ . Correct synthesis of ψ then amounts to constructing such P .

So far, we considered the case where the specifications refer solely to signals in I and O , both are known to P . This is called synthesis with *complete information*. Often, the system does not have complete information about its environment. That is, there is a set E of signals that the environment generates but the system cannot read. Since P cannot read the signals in E , its activity is independent of them. Hence, it can still be viewed as a strategy $P : (2^I)^* \rightarrow 2^O$. Nevertheless, the computations of P are now infinite words over $2^{I \cup E \cup O}$. Similarly, embodying all the possible inputs to P , the computation tree induced by P now has a fixed branching degree $|2^{I \cup E}|$ and it is labeled by letters in $2^{I \cup E \cup O}$. Note that different nodes in this tree may have, according to P 's incomplete information, the same “history of inputs” (that is, when we project the labels along the paths from the root to these nodes on $2^{I \cup O}$, we get the same computation).

Often, systems need to satisfy specifications that refer to signals they cannot read. For example, in a distributed setting, each process is an open system and it can read only part of the signals generated by the other processes. Formally, given a specification ψ over the sets I , E , and O of readable input, unreadable input, and output signals, respectively, *synthesis with incomplete information* amounts to constructing a system $P : (2^I)^* \rightarrow 2^O$, which is independent of E , and which realizes ψ (that is, if ψ is linear then all the computations of P satisfy ψ , and if ψ is branching then the computation tree of P satisfies ψ). It is known how to cope with incomplete information in the linear paradigm. In particular, the approach used in [20] can be extended to handle synthesis with incomplete information for linear specifications [13,15,23]. Coping with incomplete

¹ All along this work, we consider synthesis with respect to *maximal environments*, which provide all possible input sequences.

information is more difficult in the branching paradigm, where the methods used in the linear paradigm are not applicable [16].

In [16] we solved the problem of synthesis with incomplete information for specification in the branching temporal logic CTL^{*} and its subset CTL. We proved that independently of the presence of incomplete information, the synthesis problems for CTL^{*} and CTL are complete for 2EXPTIME and EXPTIME, respectively. These results joined the 2EXPTIME-complete bound for LTL synthesis in both settings [20,22,23]. Keeping in mind that the satisfiability problems for LTL, CTL, and CTL^{*} are complete for PSPACE, EXPTIME, and 2EXPTIME, respectively [9], it follows that while the transition from closed to open systems dramatically increases the complexity of synthesis in the linear paradigm, it does not seem to influence the complexity in the branching paradigm. In both paradigms, incompleteness of the information does not make the synthesis problem more complex.

In this work, we consider the specification language μ -calculus. The μ -calculus is a propositional modal logic augmented with least and greatest fixpoint operators. It was introduced in [14], following earlier studies of fixpoint calculi in the theory of program correctness [4]. Over the past fifteen years, the μ -calculus has been established as essentially the “ultimate” program logic, as it expressively subsumes all propositional program logics, including dynamic logics such as PDL [10], and temporal logics such as LTL and CTL^{*} [6]. The μ -calculus has gained further prominence with the discovery that its formulas can be evaluated symbolically in a natural way [2], leading to industrial acceptance of computer-aided verification.

As we explain below, the techniques developed in [16] do not extend in a straightforward manner to specification expressed in the μ -calculus. The solution described in [16] goes through alternating tree automata. In order to cope with incomplete information, we need to transform formulas to alternating tree automata that meet two structural restrictions: the automata are ε -free (i.e., they contain no ε -transitions and all the copies of the automaton that are generated during a transition are sent to successors of the node currently read in the input tree) and they are *symmetric* (i.e., the transition function of the automaton does not distinguish between different successors of a node in the input tree, and the automaton proceeds either existentially (a copy of the automaton is sent to some unspecified successor) or universally (copies of the automaton are sent to all successors)). Previous translations of μ -calculus to alternating tree automata result in automata that either contain ε -transitions or are not symmetric². Maintaining both structural requirements involves alternating automata with quadratically many states and transitions of exponential size. The exponential blow-up is not a problem when the specification language is CTL^{*} (there, it is absorbed in the overall doubly-exponential complexity), and it does not occur when the specification language is CTL (there, we know how to construct ε -free symmetric alternating automata with no exponential blow-up). As we explain below, for μ -calculus the challenge is to absorb this exponential blow-up in the exponential cost of the final algorithm, rather than have it blow-up the complexity by another exponential.

² The first translation in [7] does not contain ε -transitions, but it implicitly assumes them (otherwise, it is incorrect). The translations in [1,24,11] have ε -transitions, and the one in [17] is not symmetric.

Technically, the synthesis problem for μ -calculus is reduced to checking the nonemptiness of an alternating automaton with polynomially many states, with respect to trees whose branching degree is exponential. We can translate the alternating automaton to a nondeterministic one with an exponential blow up [18]. Since the nonemptiness of this automaton has to be checked with respect to trees whose branching degree is exponential as well, its transition relation can be of a doubly-exponential size, resulting in a doubly-exponential complexity for the synthesis problem. Using the fact that parity games has memoryless strategies [7], we are able to translate the alternating automaton to a deterministic one, running over trees annotated with winning strategies. Being deterministic, the transition relation of the automaton is of exponential size even when it runs on trees with an exponential branching degree. The key to achieving these complexity bounds is a novel, efficient encoding of winning strategies. We believe that our technique will be useful also in other contexts. For example, it implies a translation of alternating parity automata to nondeterministic ones that is exponential, independently of the branching degree of the underlying trees.

Using these automata-theoretic results, we obtain an exponential-time algorithm for the problem of μ -calculus synthesis with incomplete information, implying that the problem is EXPTIME-complete. The exact complexity of the algorithm is $2^{O(n^6)}$, where n is the length of the synthesized formula. This discouraging blow-up rarely appears in practice, and the problem of whether the exponent can be improved is left open. From a theoretical point of view, this result strengthens the observation about satisfiability and synthesis having the same complexity in the branching paradigm [7,10]. In addition, using known translations of CTL and CTL* into the μ -calculus, all the results in [16] can be obtained as a special case of our result here. Due to the lack of space, this version does not contain proofs and a detailed description of the constructions. The interested reader is referred to the full version, available at the authors' web sites.

2 Preliminaries

2.1 Trees and Labeled Trees

Given a finite set $\mathcal{Y}$, an $\mathcal{Y}$ -tree is a set $T \subseteq \mathcal{Y}^*$ such that if $x \cdot v \in T$, where $x \in \mathcal{Y}^*$ and $v \in \mathcal{Y}$, then also $x \in T$. When $\mathcal{Y}$ is not important or clear from the context, we call T a tree. The elements of T are called *nodes*, and the empty word ε is the *root* of T . For every $x \in T$, the nodes $x \cdot v \in T$, for $v \in \mathcal{Y}$, are the *children* of x . Each node x of T has a *direction* in $\mathcal{Y}$. The direction of the root is v^0 , for some designated $v^0 \in \mathcal{Y}$, called the *root direction*. The direction of a node $x \cdot v$ is v . We denote by $\text{dir}(x)$ the direction of node x . An $\mathcal{Y}$ -tree T is a *full infinite tree* if $T = \mathcal{Y}^*$. A *path* π of a tree T is a minimal set $\pi \subseteq T$ such that $\varepsilon \in \pi$ and for every $x \in \pi$ there exists a unique $v \in \mathcal{Y}$ such that $x \cdot v \in \pi$.

Given two finite sets $\mathcal{Y}$ and Σ , a Σ -labeled $\mathcal{Y}$ -tree is a pair $\langle T, V \rangle$ where T is an $\mathcal{Y}$ -tree and $V : T \rightarrow \Sigma$ maps each node of T to a letter in Σ . When $\mathcal{Y}$ and Σ are not important or clear from the context, we call $\langle T, V \rangle$ a labeled tree. For a Σ -labeled $\mathcal{Y}$ -tree $\langle T, V \rangle$, we define the *x-ray* of $\langle T, V \rangle$, denoted $\text{xray}(\langle T, V \rangle)$, as the $(\mathcal{Y} \times \Sigma)$ -labeled $\mathcal{Y}$ -tree $\langle T, V' \rangle$ in which each node is labeled by both its direction and its labeling in $\langle T, V \rangle$.

Thus, for every $x \in T$, we have $V'(x) = \langle \text{dir}(x), V(x) \rangle$. We say that a $(Y \times \Sigma)$ -labeled Y -tree $\langle T, V \rangle$ is *Y-exhaustive* if for every node $x \in T$, we have $V(x) \in \text{dir}(x) \times \Sigma$. Note that for every Σ -labeled Y -tree $\langle T, V \rangle$, the tree $\text{xray}(\langle T, V \rangle)$ is Y -exhaustive.

Let X and Y be finite sets. Consider the tree $(X \times Y)^*$. We define a function $\text{hide}_Y : (X \times Y)^* \rightarrow X^*$. Given a node $w \in (X \times Y)^*$, the node $\text{hide}_Y(w) \in X^*$ is obtained from w by replacing each letter $\langle x, y \rangle$ in w by the letter x . For example, if $X = Y = \{0, 1\}$, then the node 0010 of the 4-ary tree in the figure corresponds, by hide_Y , to the node 01 of the binary tree. Note that the nodes 0011, 0110, and 0111 of the 4-ary tree also correspond, by hide , to the node 01 of the binary tree. We extend the hiding operator to paths $\pi \subset (X \times Y)^*$ in the straightforward way. That is, the path $\text{hide}_Y(\pi) \subset X^*$ is obtained from π by replacing each node $w \in \pi$ by the node $\text{hide}_Y(w)$.

Let Z be a finite set. For a Z -labeled X -tree $\langle X^*, V \rangle$, we define the *Y-widening* of $\langle X^*, V \rangle$, denoted $\text{wide}_Y(\langle X^*, V \rangle)$, as the Z -labeled $(X \times Y)$ -tree $\langle (X \times Y)^*, V' \rangle$ where for every node $w \in (X \times Y)^*$, we have $V'(w) = V(\text{hide}_Y(w))$. Note that for every node $w \in (X \times Y)^*$ and $x \in X$, the children $w \cdot \langle x, y \rangle$, for all $y \in Y$, agree on their label in $\langle (X \times Y)^*, V' \rangle$. Indeed, they are all labeled with $V(\text{hide}_Y(w) \cdot x)$. The essence of widening is that for every path π in $\langle X^*, V \rangle$ and for every path $\rho \in \text{hide}_Y^{-1}(\pi)$, the path ρ exists in $\langle (X \times Y)^*, V' \rangle$ and $V(\pi) = V'(\rho)$.

2.2 Symmetric Automata and the Propositional μ -Calculus

Let $\Omega = \{\varepsilon, \square, \diamond\}$, and let $\mathcal{B}^+(\Omega \times Q)$ be the set of positive Boolean formulas over $\Omega \times Q$; i.e., Boolean formulas built from elements in $\Omega \times Q$ using $\wedge$ and $\vee$. For a set $S \subseteq \Omega \times Q$ and a formula $\theta \in \mathcal{B}^+(\Omega \times Q)$, we say that S *satisfies* θ iff assigning **true** to elements in S and assigning **false** to elements in $(\Omega \times Q) \setminus S$ makes θ true. Consider a set $\mathcal{Y} = \{v_1, \dots, v_n\}$ of directions. In a nondeterministic automaton $\mathcal{A}$, over labeled $\mathcal{Y}$ -trees, with a set Q of states, the transition function δ maps an automaton state $q \in Q$ and an input letter $\sigma \in \Sigma$ to a set of n -tuples of states. Each n -tuple suggests a nondeterministic choice for the automaton's next configuration. A *symmetric automaton* [12,25] is an alternating tree automaton in which the transition function δ maps q and σ to a formula in $\mathcal{B}^+(\Omega \times Q)$. Atoms of the form $\langle \varepsilon, q \rangle$ are called ε -transitions. Intuitively, an atom $\langle \varepsilon, q \rangle$ corresponds to a copy of the automaton in state q sent to the current node of the input tree. An atom $\langle \square, q \rangle$ corresponds to n copies of the automaton in state q , sent to all the successors of the current node. An atom $\langle \diamond, q \rangle$ corresponds to a copy of the automaton in state q , sent to some successor of the current node.

Formally, a symmetric automaton is a tuple $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$ where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \rightarrow \mathcal{B}^+(\Omega \times Q)$ is a transition function, $q_0 \in Q$ is an initial state, and α specifies the acceptance condition (a condition that defines a subset of Q^ω). The automaton $\mathcal{A}$ is ε -free iff δ contains no ε -transitions. A *run* of a symmetric automaton $\mathcal{A}$ on an input labeled $\mathcal{Y}$ -tree $\langle T, V \rangle$ is a tree $\langle T_r, r \rangle$ (to be formally defined shortly) in which each node is labeled by an element of $\mathcal{Y}^* \times Q$. Unlike T , in which each node has exactly $|\mathcal{Y}|$ children, the tree T_r may have nodes with many children and may also have *leaves* (nodes with no children). Thus, $T_r \subset \mathbb{N}^*$ and a path in T_r may be either finite, in which case it ends in a leaf, or infinite. Each node of T_r corresponds to a node of T . A node in T_r , labeled by (x, q) , describes a copy of

the automaton that reads the node x of T and visits the state q . Note that many nodes of T_r can correspond to the same node of T ; in contrast, in a run of a nondeterministic automaton on $\langle T, V \rangle$ there is a one-to-one correspondence between the nodes of the run and the nodes of the tree. The labels of a node and its children have to satisfy the transition function. Formally, the run $\langle T_r, r \rangle$ is an $(T^* \times Q)$ -labeled tree such that $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$, and for all $y \in T_r$ with $r(y) = (x, q)$ and $\delta(q, V(x)) = \theta$, there is a (possibly empty) set $S \subseteq \Omega \times Q$, such that S satisfies θ , and for all $(c, s) \in S$, the following hold:

- If $c = \varepsilon$, then there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x, s)$.
- If $s = \square$, then for each $v \in \mathcal{T}$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot v, s)$.
- If $s = \diamond$, then for some $v \in \mathcal{T}$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot v, s)$.

Each infinite path ρ in $\langle T_r, r \rangle$ is labeled by a word in Q^ω . Let $\text{inf}(\rho)$ denote the set of states in Q that appear in $r(\rho)$ infinitely often. A run $\langle T_r, r \rangle$ is accepting iff all its infinite paths satisfy the acceptance condition. In *parity* automata, α is a partition $\{F_1, F_2, \dots, F_k\}$ of Q and an infinite path ρ satisfies α iff the minimal index i for which $\text{inf}(\rho) \cap F_i \neq \emptyset$ is even. *Co-parity* automata are similar, only that the minimal index i for which $\text{inf}(\rho) \cap F_i \neq \emptyset$ is odd. The number k is called the *index* of the automaton. A *safety* automaton is a parity automaton with $\alpha = \{\emptyset, Q\}$. Thus, all the words in Q^ω satisfy α , and we do not specify it.

An automaton accepts a tree iff there exists an accepting run on it. We denote by $\mathcal{L}(\mathcal{A})$ the language of the automaton $\mathcal{A}$; i.e., the set of all labeled trees that $\mathcal{A}$ accepts. We say that $\mathcal{A}$ is *nonempty* iff $\mathcal{L}(\mathcal{A}) \neq \emptyset$. We denote by $\mathcal{A}^q$ the automaton obtained from $\mathcal{A}$ by making q the initial state.

The *propositional μ -calculus* is a propositional modal logic augmented with least and greatest fixpoint operators [14]. Formulas of μ -calculus can be translated to symmetric parity automata with no blow up:

Theorem 1. [17] *Given a μ -calculus formula ψ , we can construct a symmetric parity automaton $\mathcal{A}_\psi$ such that $\mathcal{L}(\mathcal{A}_\psi)$ is exactly the set of trees satisfying ψ . The automaton $\mathcal{A}_\psi$ has $|\psi|$ states and index at most $|\psi|$.*

Theorem 1 enables us to solve the problem of μ -calculus synthesis by means of symmetric parity automata (that is, assuming that the specification is given as a symmetric parity automaton)³.

3 The Problem

Consider a system P that interacts with its environment. Let I and E be the sets of input signals, readable and unreadable by P , respectively, and let O be the set of P 's

³ This means that readers not familiar with μ -calculus can continue to read the paper assuming it studies synthesis of specifications given by symmetric parity automata, keeping in mind that natural applications would start with a μ -calculus formula, automatically transformed to such an automaton.

output signals. We can view P as a *strategy* $P : (2^I)^* \rightarrow 2^O$. Indeed, P maps each finite sequence of sets of readable input signals into a set of output signals. Note that the information available to P regarding its environment is incomplete and P does not depend on the unreadable signals in E . We assume the following interaction between P and its environment. The interaction starts by P outputting $P(\varepsilon)$. The environment replies with some $\langle i_1, e_1 \rangle \in 2^I \times 2^E$, to which P replies with $P(i_1)$. Interaction then continues step by step, with an output $P(i_1 \cdot i_2 \cdots i_j)$ corresponding to a sequence $\langle i_1, e_1 \rangle \cdot \langle i_2, e_2 \rangle \cdots \langle i_j, e_j \rangle$ of inputs. Thus, P associates with each infinite input sequence $\langle i_1, e_1 \rangle \cdot \langle i_2, e_2 \rangle \cdots$, an infinite computation $[P(\varepsilon)] \cdot [i_1 \cup e_1 \cup P(i_1)] \cdot [i_2 \cup e_2 \cup P(i_1 \cdot i_2)] \cdots$ over $2^{I \cup E \cup O}$. We note that our choice of P starting the interaction is for technical convenience only.

Though the system P is deterministic, it induces a *computation tree*. The branches of the tree correspond to external nondeterminism, caused by different possible inputs. Thus, $tree(P)$ is a $2^{I \cup E \cup O}$ -labeled $2^{I \cup E}$ -tree, where each node with direction $i \cup e \in 2^I \cup 2^E$ is labeled by $i \cup e \cup o$, where o is the set of output signals that P assigns to the sequence of readable inputs leading to the node. Formally, we obtain the computation tree $tree(P)$ by two transformations on the 2^O -labeled tree $\langle (2^I)^*, P \rangle$, which represents P . First, while $\langle (2^I)^*, P \rangle$ ignores the signals in E and the extra external nondeterminism induced by them, the computation tree of P , which embodies all possible computations, takes them into account. For that, we define the 2^O -labeled tree $\langle (2^{I \cup E})^*, P' \rangle = wide_{(2^E)}(\langle (2^I)^*, P \rangle)$. By the definition of the *wide* operator, each two nodes in $(2^{I \cup E})^*$ that correspond, according to P 's incomplete information, to the same input sequence are labeled by P' with the same output. Now, as the signals in I and E are represented in $\langle (2^{I \cup E})^*, P' \rangle$ only in its nodes and not in its labels, we define the computation tree $tree(P)$ as $\langle (2^{I \cup E})^*, P'' \rangle = xray(\langle (2^{I \cup E})^*, P' \rangle)$. Note that, as I , E , and O are disjoint, we refer to $wide_{(2^E)}(\langle (2^I)^*, P \rangle)$ as a $2^{I \cup E}$ -tree, rather than a $(2^I \times 2^E)$ -tree. Similarly, $xray(\langle (2^{I \cup E})^*, P' \rangle)$ is a $2^{I \cup E \cup O}$ -labeled tree, rather than a $(2^{I \cup E} \times 2^O)$ -labeled tree.

Given a μ -calculus formula ψ over the sets $I \cup E \cup O$ of atomic propositions, the problem of *realizability with incomplete information* is to determine whether there is a system P whose computation tree $tree(P)$ satisfies ψ . The *synthesis problem* requires the construction of such P .

4 μ -Calculus Synthesis

Often, it is convenient to specify properties with automata containing ε -transitions. In particular, the translation of μ -calculus to symmetric automata results in automata that contain ε -transitions. As we shall see, for synthesis with incomplete information, we need automata that are both symmetric and ε -free. This is taken care of in the following theorem.

Theorem 2. [24,25] *Given a symmetric parity automaton $\mathcal{A}$, we can construct an equivalent ε -free symmetric parity automaton $\mathcal{A}'$. If $\mathcal{A}$ has n states and index k , then $\mathcal{A}'$ has $O(n^2)$ state and index $O(k)$.*

Note that while Theorem 2 provides a quadratic upper bound on the number of states in $\mathcal{A}'$, it does not provide a polynomial bound on the size of $\mathcal{A}'$. In particular, the length of the transitions of $\mathcal{A}'$ could be exponential in n . We denote $\mathcal{A}'$ by $\text{free}(\mathcal{A})$.

Recall that the operator wide_Y maps a Z -labeled X -tree $\langle X^*, V \rangle$ to a Z -labeled $(X \times Y)$ -tree $\langle (X \times Y)^*, V' \rangle$ such that for every node $w \in (X \times Y)^*$, we have $V'(w) = V(\text{hide}_Y(w))$. We define a variant of the operator wide_Y , called fat_Y . Given a Z -labeled X -tree $\langle X^*, V \rangle$, the operator fat_Y maps $\langle X^*, V \rangle$ into a set of $(Z \times Y)$ -labeled $(X \times Y)$ -trees such that $\langle (X \times Y)^*, V' \rangle \in \text{fat}_Y(\langle X^*, V \rangle)$ iff $V'(\varepsilon) \in \{V(\varepsilon)\} \times Y$, and for every $w \in (X \times Y)^+$ with $\text{dir}(w) \in X \times \{y\}$, we have $V'(w) = \langle V(\text{hide}_Y(w)), y \rangle$. That is, $\text{fat}_Y(\langle X^*, V \rangle)$ contains $|Y|$ trees, which differ only on the label of their roots. The trees in $\text{fat}_Y(\langle X^*, V \rangle)$ are very similar to the tree $\text{wide}_Y(\langle X^*, V \rangle)$, with each node labeled, in addition to its label in $\text{wide}_Y(\langle X^*, V \rangle)$, also with the Y -element of its direction. An exception is the root, which is labeled, in addition to its label in $\text{wide}_Y(\langle X^*, V \rangle)$, also with some element of Y . Among all the trees in $\text{fat}_Y(\langle X^*, V \rangle)$, of special interest to us is the tree with root labeled with $\langle V(\varepsilon), y^0 \rangle$, where y^0 is the root direction of Y . We call this tree $\text{wide}'_Y(\langle X^*, V \rangle)$.

Theorem 3. *Let X, Y , and Z be finite sets. Given an ε -free symmetric parity automaton $\mathcal{A}$, over $(Z \times Y)$ -labeled $(X \times Y)$ -trees, we can construct an ε -free symmetric parity automaton $\mathcal{A}'$ over Z -labeled X -trees such that $\mathcal{A}'$ accepts a Z -labeled tree $\langle X^*, V \rangle$ iff $\mathcal{A}$ accepts the $(Z \times Y)$ -labeled tree $\text{wide}'_Y(\langle X^*, V \rangle)$. If $\mathcal{A}$ has n states and index k , then $\mathcal{A}'$ has $O(n)$ states and index k .*

Typically, the state space of $\mathcal{A}'$ is $Q \times \{\forall, \exists\}$. When $\mathcal{A}'$ is in state $\langle q, \forall \rangle$, it accepts a tree $\langle T, V \rangle$ if all the trees in $\text{fat}_Y(\langle T, V \rangle)$ are accepted by $\mathcal{A}^q$. When the automaton is in state $\langle q, \exists \rangle$, it accepts all trees $\langle T, V \rangle$ for which there exists a tree in $\text{fat}_Y(\langle T, V \rangle)$ that is accepted by $\mathcal{A}^q$. In the full version, we describe the construction in detail and we explain why the ε -freeness of $\mathcal{A}$ is essential. We denote $\mathcal{A}'$ by $\text{narrow}_Y(\mathcal{A})$.

Consider an ε -free symmetric automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$. Recall that the transition function δ maps a state and a letter to a formula in $\mathcal{B}^+(\{\square, \diamond\} \times Q)$. For a set $\mathcal{T}$ of directions, let $\Psi_{\mathcal{T}} = (\{\square\} \cup \mathcal{T}) \times Q$. A subset H of $\Psi_{\mathcal{T}}$ can be viewed as an “detailed” description of a set of atoms in $\{\square, \diamond\} \times Q$, where instead an atom $(\diamond, q)$, the set H contains an atom (v, q) , where $v \in \mathcal{T}$ is the direction in which the existential requirement $(\diamond, q)$ is going to be satisfied. We say that H satisfies a formula θ in $\mathcal{B}^+(\{\square, \diamond\} \times Q)$ if the set obtained from H by replacing all the atoms in $\mathcal{T} \times Q$ by the corresponding atoms in $\{\diamond\} \times Q$ satisfies θ . A restriction of δ is a function $\delta' : Q \times \Sigma \rightarrow 2^{\Psi_{\mathcal{T}}}$ such that for all $q \in Q$ and $\sigma \in \Sigma$ for which $\delta(q, \sigma)$ is satisfiable (i.e., $\delta(q, \sigma)$ is not **false**), the set $\delta'(q, \sigma)$ satisfies $\delta(q, \sigma)$. If $\delta(q, \sigma)$ is not satisfiable, then $\delta'(q, \sigma)$ is undefined. Let F_{δ} be the set of restrictions of δ .

Consider a Σ -labeled tree $\langle \mathcal{T}^*, V \rangle$. A running strategy of $\mathcal{A}$ for $\langle \mathcal{T}^*, V \rangle$ is an F_{δ} -labeled tree $\langle \mathcal{T}^*, f \rangle$. The running strategy $\langle \mathcal{T}^*, f \rangle$ induces a single run $\langle T_r, r_f \rangle$ of $\mathcal{A}$ on $\langle \mathcal{T}^*, V \rangle$. Intuitively, whenever the run $\langle T_r, r_f \rangle$ is in state q as it reads a node $x \in \mathcal{T}^*$, it proceeds according to $f(x)(q, V(x))$. Formally, $\langle T_r, r_f \rangle$ is the $(\mathcal{T}^* \times Q)$ -labeled tree such that $\varepsilon \in T_r$ and $r(\varepsilon) = (\varepsilon, q_0)$, and for all $y \in T_r$ with $r(y) = (x, q)$, the function $f(x)(q, V(x))$ is defined and for all $(c, s) \in f(x)(q, V(x))$, the following hold:

- If $c = \square$, then for each $v \in \Upsilon$, there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot v, s)$.
- If $c \in \Upsilon$, then there is $j \in \mathbb{N}$ such that $y \cdot j \in T_r$ and $r(y \cdot j) = (x \cdot c, s)$.

For a node $x \in \Upsilon^*$ and a state $q \in Q$, we say that x is *obliged to* q by f and V if $x = \varepsilon$ and $q = q_0$, or $x = y \cdot v$, for $y \in \Upsilon^*$ and $v \in \Upsilon$, and there is a state q' such that y is obliged to q' and $f(y)(q', V(y))$ contains $(\square, q)$ or (v, q) . Thus, x is obliged to q if x is visited by q in the run $\langle T_r, r_f \rangle$.

A *will* for the automaton $\mathcal{A}$ is a function $\rho : Q \rightarrow 2^Q$. Let G_Q be the set of all wills. A *promise* of $\mathcal{A}$ for a Σ -labeled tree $\langle \Upsilon^*, V \rangle$ is a G_Q -labeled tree $\langle \Upsilon^*, g \rangle$. Intuitively, the promise $\langle \Upsilon^*, g \rangle$ corresponds to a run of $\mathcal{A}$ on $\langle \Upsilon^*, V \rangle$ in which for all nodes $y \cdot v$ and states $q \in Q$, if y is visited by state q , then $y \cdot v$ is visited by all the states in $g(y \cdot v)(q)$. For an infinite sequence $\rho_0, \rho_1, \dots$ of wills for $\mathcal{A}$ and sequence (either finite or infinite) $q_0, q_1, \dots$ of states, we say that $q_0, q_1, \dots$ is a *trace* induced by $\rho_0, \rho_1, \dots$ if q_0 is the initial state of $\mathcal{A}$ and for every $i \geq 0$, either $\rho_{i+1}(q_i)$ is empty, in which case q_i is the last state in the trace, or $\rho_{i+1}(q_i)$ is not empty, in which case q_{i+1} belongs to $\rho_{i+1}(q_i)$. Note that the trace is independent of ρ_0 . We say that a promise $\langle \Upsilon^*, g \rangle$ is *good* for $\mathcal{A}$ if all the infinite traces induced by paths in $\langle \Upsilon^*, g \rangle$ satisfy the acceptance condition α .

Consider a Σ -labeled tree $\langle \Upsilon^*, V \rangle$, a running strategy $\langle \Upsilon^*, f \rangle$, and a promise $\langle \Upsilon^*, g \rangle$. We say that g *fulfills* f for V if the states promised to be visited by g satisfy the obligations induced by f as it runs on V . Formally, g fulfills f for V if for every node $x \in \Upsilon^*$ and state q such that x is obliged to q by f and V , the following hold:

1. For every atom $(\square, s) \in f(x)(q, V(x))$, all the successors $x \cdot v$ of x have $s \in g(x \cdot v)(q)$.
2. For every atom $(v, s) \in f(x)(q, V(x))$, the successor $x \cdot v$ of x has $s \in g(x \cdot v)(q)$.

Theorem 4. *$\mathcal{A}$ accepts $\langle \Upsilon^*, V \rangle$ iff there exist a running strategy $\langle \Upsilon^*, f \rangle$ and a promise $\langle \Upsilon^*, g \rangle$ such that $\langle \Upsilon^*, g \rangle$ is good for $\mathcal{A}$ and g fulfills f for V .*

Annotating input trees with restrictions and wills enables us to transform a symmetric automaton to a deterministic one, with an exponential blow up:

Theorem 5. *Consider an ε -free symmetric parity automaton $\mathcal{A}$ such that $\mathcal{A}$ runs on Σ -labeled Υ -trees. There is a deterministic parity tree automaton $\mathcal{A}'$ such that $\mathcal{A}'$ accepts a $(\Sigma \times F_\delta \times G_Q)$ -labeled Υ -tree iff $\mathcal{A}$ accepts its projection on Σ . If $\mathcal{A}$ has n states and index k , then $\mathcal{A}'$ has $2^{n(n+k \log n k)}$ states and index nk .*

We refer to a Σ' -labeled Υ -tree as $\langle \Upsilon^*, (V, f, g) \rangle$, where V, f , and g are the projections of the tree on Σ, F_δ , and G_Q , respectively. Typically, the automaton $\mathcal{A}'$ is the intersection of two deterministic automata $\mathcal{A}'_1$ and $\mathcal{A}'_2$. The automaton $\mathcal{A}'_1$ accepts a tree $\langle \Upsilon^*, (V, f, g) \rangle$ iff $\langle \Upsilon^*, g \rangle$ is a good promise for $\mathcal{A}$. The automaton $\mathcal{A}'_2$ accepts a tree $\langle \Upsilon^*, (V, f, g) \rangle$ iff g fulfills f for V . By Theorem 4, it follows that $\mathcal{A}'$ accepts $\langle \Upsilon^*, (V, f, g) \rangle$ iff $\mathcal{A}$ accepts $\langle \Upsilon^*, V \rangle$. We call $\mathcal{A}'$ the *witnessing* automaton for $\mathcal{A}$, and denote it by $\text{witness}(\mathcal{A})$. Finally, we need the following construction, which checks an $(\Upsilon \times \Sigma)$ -labeled Υ -tree for being Υ -*exhaustive*.

Theorem 6. [16] *Given finite sets Υ and Σ , there is a deterministic safety tree automaton $\mathcal{A}_{exh}$ on $(\Upsilon \times \Sigma)$ -labeled Υ -trees, with $|\Upsilon|$ states, such that $\mathcal{L}(\mathcal{A}_{exh})$ is exactly the set of Υ -exhaustive trees. All the states of $\mathcal{A}_{exh}$ are accepting.*

Given a specification ψ of length n , let $\mathcal{A} = \text{witness}(\text{narrows}(2^E)(\text{free}(\mathcal{A}_\psi)))$. By Theorems 1, 2, and 3, the automaton $\mathcal{A}$ has $2^{O(n^4)}$ states and index $O(n^2)$, and the specification ψ is realizable iff $\mathcal{L}(\mathcal{A})$ contains a 2^I -exhaustive tree. Hence, the specification ψ is realizable iff $\mathcal{A} \times \mathcal{A}_{exh}$ is not empty. The nonemptiness problem for a deterministic parity automaton with n states and index k can be solved in time $O(n^k)$ [8], which implies a $2^{O(n^6)}$ complexity for the realizability problem. The nonemptiness algorithm can be extended, within the same complexity bound, to generate a finite-state strategy whose tree is accepted by the automaton [21], thus solving the synthesis problem. Hence the following theorem.

Theorem 7. *The synthesis problem for μ -calculus, with either complete or incomplete information, is EXPTIME-complete.*

References

1. G. Bhat and R. Cleaveland. Efficient local model-checking for fragments of the modal μ -calculus. In *Proc. TACAS*, LNCS 1055, 1996.
2. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information & Computation*, 98(2):142–170, 1992.
3. M. Daniele, P. Traverso, and M.Y. Vardi. Strong cyclic planning revisited. In *Proc 5th European Conference on Planning*, pp. 34–46, 1999.
4. E.A. Emerson and E.M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proc. 7th ICALP*, pp. 169–181, 1980.
5. E.A. Emerson and E.M. Clarke. Using branching time logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.
6. E.A. Emerson and J.Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *Journal of the ACM*, 33(1):151–178, 1986.
7. E.A. Emerson and C. Jutla. Tree automata, μ -calculus and determinacy. In *Proc. 32nd FOCS*, pp. 368–377, 1991.
8. E.A. Emerson, C. Jutla, and A.P. Sistla. On model-checking for fragments of μ -calculus. In *Proc. 5th CAV*, LNCS 697, pp. 385–396, 1993.
9. E.A. Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science*, pp. 997–1072, 1990.
10. M.J. Fischer and R.E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and Systems Sciences*, 18:194–211, 1979.
11. E. Graedel and I. Walukiewicz. Guarded fixed point logic. In *Proc. 14th LICS*, 1999.
12. D. Janin and I. Walukiewicz. Automata for the modal μ -calculus and related results. In *Proc. 20th MFCS*, LNCS, pp. 552–562, 1995.
13. R. Kumar and V.K. Garg. *Modeling and control of logical discrete event systems*. Kluwer Academic Publishers, 1995.
14. D. Kozen. Results on the propositional μ -calculus. *Theoretical Computer Science*, 27:333–354, 1983.
15. R. Kumar and M.A. Shayman. Supervisory control of nondeterministic systems under partial observation and decentralization. *SIAM J. of Control and Optimization*, 1995.

16. O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *Proc. 2nd ICTL*, pp. 91–106, July 1997.
17. O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2), March 2000.
18. D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.
19. Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM TOPLAS*, 2(1):90–121, 1980.
20. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *16th POPL*, 1989.
21. M.O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Math. Logic and Foundations of Set Theory*, pp. 1–23. North Holland, 1970.
22. R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, Rehovot, Israel, 1992.
23. M.Y. Vardi. An automata-theoretic approach to fair realizability and synthesis. In *Proc. 7th CAV*, LNCS 939, pp. 267–292, 1995.
24. M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th ICALP*, LNCS 1443, pp. 628–641, 1998.
25. T. Wilke. CTL^+ is exponentially more succinct than CTL. In *Proc. 19th TST & TCS*, LNCS 1738, pp. 110–121, 1999.

The Infinite Versions of $\text{LOGSPACE} \neq \text{P}$ Are Consistent with the Axioms of Set Theory

Grégory Lafitte and Jacques Mazoyer

Ecole Normale Supérieure de Lyon,
Laboratoire de l'Informatique du Parallélisme,
46 allée d'Italie, 69364 Lyon Cedex 07, France
{glafitte, mazoyer}@ens-lyon.fr

Abstract. We consider the infinite versions of the usual computational complexity questions $\text{LOGSPACE} \stackrel{?}{=} \text{P}$, $\text{NLOGSPACE} \stackrel{?}{=} \text{P}$ by studying the comparison of their descriptive logics on infinite partially ordered structures rather than restricting ourselves to finite structures. We show that the infinite versions of those famous class separation questions are consistent with the axioms of set theory and we give a sufficient condition on the complexity classes in order to get other such relative consistency results.

Introduction

Looking at infinite versions of problems is an approach to solving problems in complexity theory : the infinite case might be easier to solve. It is then perhaps possible to apply the proof techniques from the infinite case to the finite complexity theory questions. In one of the best examples of this technique, Sipser [15] showed that an infinite version of parity does not have bounded depth countable-size circuits using ideas from descriptive set theory. By making an analogy between *polynomial* and *countable*, Furst, Saxe and Sipser [5] used the techniques of the infinite case from Sipser's paper to show that parity does not have constant-depth polynomial-size circuits.

Recall that in descriptive complexity, two logically characterizable complexity classes $\mathcal{C}$, $\mathcal{C}'$ are equal ($\mathcal{C} = \mathcal{C}'$) if and only if the corresponding logics $\mathcal{L}_{\mathcal{C}}$ and $\mathcal{L}_{\mathcal{C}'}$ correspond¹ on ordered finite structures. Our study focuses on the comparison of the logics on partially ordered infinite structures. This is what we call the infinite version of complexity class separation questions.

We settle for the infinite case the usual computational complexity question in an unusual way : “the infinite version of $(\text{N})\text{LOGSPACE} \neq \text{P}$ ” is consistent with the standard axioms of set theory. Apart from the trivial separation of Σ_1^1 and Π_1^1 (NP and co-NP) on structures of cardinality $\kappa \geq \omega$, little was known. Fortnow, Kurtz and Whang [4] pointed out an open communication complexity

This work was partially supported by a Région Rhône-Alpes EURODOC France-Israel grant.

¹ Each sentence in $\mathcal{L}_{\mathcal{C}}$ has an equivalent sentence (same models) in $\mathcal{L}_{\mathcal{C}'}$ and vice versa.

problem whose infinite version Miller [13] had proved to be independent. As far as we know, our results are the first known relative consistency results for infinite versions of complexity questions, for which the infinite versions were not directly connected to already known relative consistent propositions in set theory.

Note that the relative consistency of the infinite case does not imply anything about the provability of the usual computational complexity questions. What it does tell us, is that any proof that LOGSPACE and P correspond (in the usual finite case meaning) will not carry over to the infinite case.

As indicated above, the infinite separation of NP and co-NP is straight forward : it is a known fact from set-theoretical absoluteness study that Σ_1^1 and Π_1^1 separate on infinite structures, e.g. “ α is an ordinal”. Of course, the separation of NP and co-NP on infinite structures also implies, because P and PSPACE are closed by complementary, the separation of P and NP, P and co-NP, NP and PSPACE, co-NP and PSPACE, and obviously also the separation of classes contained in P and of classes containing NP or co-NP such as LOGSPACE and NP (co-NP), P and PSPACE. Thus the only non-trivial case (when considering the combinations of the above mentioned complexity classes) on infinite structures is in comparing (N)LOGSPACE and P.

The complexity classes that are appropriate for our method are those that verify certain conditions, named $(\star)$, which are given in the following section. The two complexity classes $\mathcal{C}$ and $\mathcal{C}'$ must be inclusion-wise comparable (hence NP and co-NP are not appropriate), the lower complexity class $\mathcal{C}$ must be logically characterizable on ordered finite structures and $\mathcal{C}'$ must be characterizable in a certain precise way (as MONOTONE-FO[OPERATOR] also on ordered finite structures), that most of the usual computational complexity classes verify. Moreover, the two fixed point logics on first order definable functions must separate on finite structures (that are not necessarily linearly ordered). (N)LOGSPACE and P verify those $(\star)$ conditions as well as all of the above complexity classes (apart from co-NP for which we do not know). Of course, there is nothing surprising in obtaining the relative consistency of a provable proposition. In the rest of the paper, we strive to show the relative consistency of the separation for classes verifying the conditions detailed above and not only for (N)LOGSPACE and P.

Note that the result is indeed about the consistency of the separation of *all* infinite versions of the complexity classes that verify certain precise conditions. One of those is that the greater complexity class be logically characterizable by some fixed point operator on functions that are monotone for inclusion. Any infinite version of that fixed point operator is suitable as long as we keep the restriction to monotone functions. And so, we not only prove the consistency of the separation for fixed infinite versions of suitable complexity classes but mainly the consistency of the separation of all the suitable infinite interpretations. When we fix the infinite interpretation and obtain the consistency result, most of the time (at least for easy interpretations that naturally come to mind) the separation is provable and not only relatively consistent. Nevertheless it is quite surprising that in *all* cases, the separation is consistent. Those results are thus particularly interesting in the prospect of obtaining some transfer theorems

between certain infinite and finite propositions, which could then give us hints on the finite usual case.

For the proof, we fix $\mathcal{C}, \mathcal{C}'$ that verify our conditions. We then compare $\text{MONOTONE}[\text{OPERATOR}]$ and $\mathcal{C}$. We show that if the two logics correspond on certain partially ordered infinite structures, then the cardinality of those structures is an inaccessible² cardinal. This means that on structures of any infinite cardinality apart (perhaps) from inaccessible cardinals, the logics separate. We thus compare logics on structures of a certain cardinality or greater. This is known to be the same³ as comparing them on any structure.

Forcing arguments tell us that it is consistent with set theory, adjoined with some set-theoretical assumptions and “there is no inaccessible cardinal”, that every monotone (according to a particular convenient partial order) function be definable in first order logic. Hence we get the consistency of the separation of our complexity classes on infinite partially ordered structures with the axioms of set theory.

Considering large cardinals in this context is the crucial point in reaching consistency results. It had already been considered in the study of precise fixed point operators on arbitrary structure. When one has an operator function Γ on subsets of a *countable* set X , the closure ordinal $|\Gamma|$ is the smallest ordinal α such that $A_{\alpha+1} = \Gamma(A_\alpha)$ where A starts with $\emptyset$, on which we transfinitely iterate Γ . If we have C , a set of operators of a certain form on $P(X)$, then $|C| = \sup\{|\Gamma| : \Gamma \in C\}$. Gandy (unpublished) has first shown that $|\Pi_1^0| = \omega_1$ (operators that are definable by a Π_1^0 formula). Then Richter [14] obtained characterizations of certain natural extensions of Π_1^0 in terms of recursive analogues of large cardinals. In particular, it was shown that even $|\Pi_2^0|$ is much larger than the first recursively Mahlo ordinal, the first recursively hyper-Mahlo ordinal, etc. Aczel and Richter proved some recursive analogues of large cardinal characterizations of $|\Pi_n^0|$ and of $|\Delta_1^1|$, $|\Pi_1^1|$ and $|\Sigma_1^1|$. So it was predictable that we could perhaps obtain large cardinals when considering fixed point operators on subsets of *non-countable* sets. But again, the fact that we obtain large cardinals comes mostly from considering *all* fixed point operators with little requirements on their forms. The more important requirement of being definable is taken care of afterwards, by forcing techniques.

1 Appropriate Complexity Classes

The following definitions remind us to which logic a complexity class corresponds and vice versa. To get a much better overview of the descriptive study of complexity through finite model theory, the reader is invited to consult [2].

² This is one example of a large cardinal. Large cardinals are not only unbelievably greater than any cardinal you could possibly think of, but their very existence is also not provable in set theory. See Kanamori and Magidor [12,11] for a background on large cardinals, their properties and relative consistency strengths.

³ Let $\mathcal{K}$ be a class of ordered structures. $\mathcal{K}_m = \{A \in \mathcal{K} \mid |A| \geq m\}$. For any usual complexity class $\mathcal{C}$, $\mathcal{K} \in \mathcal{C}$ iff $\mathcal{K}_m \in \mathcal{C}$.

Let us recall some descriptive complexity results. The computational complexity class P is logically characterized by first order logic enhanced by a fixed point operator. A fixed point operator takes a first-order definable⁴ function, called *operator function*, F on 2^{domain} and gives a new relation $[\text{Op } F]$ such that⁵ $[\text{Op } F]\bar{x}$ if and only if $\bar{x}$ belongs to the fixed point (if it exists, otherwise $\emptyset$) of the iteration of F starting from $\emptyset$ ($\emptyset, F(\emptyset), F(F(\emptyset)), \dots$). To ensure that we have a fixed point, we can oblige the fixed point operator function to be inductive (that is $X \subseteq F(X)$) and this can easily be done by transforming the formula φ used to define the operator function to $X\bar{x} \vee \varphi(\bar{x}, X)$. First order logic enhanced by this operator is called the inductive fixed point (IFP) logic, which characterizes P on finite linearly ordered structures.

Gurevich and Shelah [8] have shown that this logic is equivalent to first order logic enhanced by least fixed points, which gives for an F (derived again from φ as above) its least (according to the subset order) fixed point starting from $\emptyset$. It has been shown that those fixed points are obtained by monotone functions (for all X, Y subsets of the domain, $X \subseteq Y$ implies $F(X) \subseteq F(Y)$), which are described by formulas *positive* in X (loosely speaking, there is an even number of $\neg$ before each occurrence of X). This is the logical description of P that we use.

Abiteboul, Vardi and Vianu [1, 2.2] found similar logical descriptions for NP and PSPACE with the use of inflationary fixed point operators. By using Gurevich's and Shelah's techniques in [8], it seems straight forward to show from those later logical descriptions that NP and PSPACE are characterizable by fixed point logics on functions that are monotone (for $\subseteq$) and definable in first order logic.

Another fixed point operator that can be defined is the (non-deterministic) *transitive closure* operator, which (over first order logic) captures the complexity class (N)LOGSPACE (see [2]).

P, NP and PSPACE are thus characterizable by fixed point logics on functions that are monotone (for $\subseteq$) and definable in first order logic. We use the notation MONOTONE – FO[OPERATOR] for such logics, where OPERATOR is different depending on the complexity class we are characterizing. There may be multiple fixed point operators *extending* MONOTONE – FO (as it is the case for NP and PSPACE), but it does not affect our study.

Definition 1. *Working on an ordered structure $\mathcal{A}$, we consider functions from $2^{\mathcal{A}^k}$ to $2^{\mathcal{A}^k}$. We say that such a function F is :*

- monotone if for all $X \subseteq Y$, $F(X) \subseteq F(Y)$;
- definable if there is a first order formula $\varphi(x_1, \dots, x_k, \bar{u}, X, \bar{Y})$ such that $F(R) = \{(a_1, \dots, a_k) \mid \mathcal{A} \models \varphi[a_1, \dots, a_k, \bar{b}, R, \bar{S}]\}$, where $\bar{b}$ and $\bar{S}$ are interpretations of $\bar{u}$ and $\bar{Y}$.

⁴ F is definable if there is a first-order formula φ , with free first-order variables $x_1, x_2, \dots, x_k$ (noted $\bar{x}$) and second-order variable X , such that $F(X)$ is the set of $\bar{x}$ which verify $\varphi(\bar{x}, X)$.

⁵ $\bar{x}$ is $x_1, \dots, x_k$ for a certain k .

We can now define the following logics : $(\text{MONOTONE} -)\text{FO}[\text{OPERATOR}]$ contains first order logic and is closed under operation OPERATOR (which to every function F assigns $\text{OPERATOR}[F]$, a k -ary relation on $\mathcal{A}$) on (monotone) definable functions : $\mathcal{A} \models [\text{OPERATOR } F] \bar{t}[\bar{b}]$ if and only if $(t_1[\bar{b}], \dots, t_k[\bar{b}]) \in \text{OPERATOR}[F]$.

To be able to go through the following sections, the two complexity classes, $\mathcal{C}$ and $\mathcal{C}'$ containing $\mathcal{C}$, that we compare should verify the three following conditions :

- $\mathcal{C}$ is characterizable by a $\text{FO}[\text{OPERATOR}^{\textcircled{1}}]$ logic;
- ($\star$) – $\mathcal{C}'$ is characterizable by a $\text{MONOTONE} - \text{FO}[\text{OPERATOR}^{\textcircled{2}}]$ logic;
- $\text{FO}[\text{OPERATOR}^{\textcircled{1}}] < \text{FO}[\text{OPERATOR}^{\textcircled{2}}]$ on finite (not necessarily ordered) structures.

Hence apart from all known, thus relatively consistent, infinite separation results such as $\text{NP} \neq \text{co-NP}$, the following are interesting possible computational class combinations : $\text{LOGSPACE} \subseteq \text{P}$ and $\text{NLOGSPACE} \subseteq \text{P}$. The third condition is a known result for $(\text{N})\text{LOGSPACE}$ and P in finite model theory (see [2][7.6.22]).

In the following, we compare, as indicated previously, the infinite versions of the logics behind usual complexity classes that verify the above ($\star$) conditions and therefore we do not talk anymore about the complexity classes themselves.

2 $\mu(L)$, $\nu(L)$ and Strong Limit Cardinals

Our goal is to show that “ $\text{MONOTONE} - \text{FO}[\text{OPERATOR}^{\textcircled{2}}]$ is not equal to a certain class $\mathcal{C}$ on infinite structures” is consistent with ZFC. To begin with, we are going to show that if $\text{MONOTONE}[\text{OPERATOR}^{\textcircled{2}}]$ and $\mathcal{C}$ correspond on κ -structures (structures of cardinality κ), then κ is a strong inaccessible cardinal. MONOTONE on a structure $\mathcal{A}$ is the class of functions from 2^{A^k} to 2^{A^k} which are monotone (not necessarily definable).

Let $\mathcal{L} = (L, \leq, \dots)$ be a partially ordered structure. We take the structure $\mathcal{A} = (A, \preceq, \dots)$ to be the power set of $\mathcal{L}$, with $\preceq$ being defined as a suitable combination of $\leq$ (pointwise) and $\subseteq$ (we will precisely define $\preceq$ when we come to ν' later on). We can then use $\mathcal{A}$ to shift the comparison of the logics to the comparison of the functions used in the logics' fixed point operators.

A structure, where every *monotone* function with an $\text{OPERATOR}^{\textcircled{2}}$ operator is equivalent to a formula in $\mathcal{C}$, will be called a *nice* structure.

For a reminder of common set theoretical definitions, consult [10].

The study of μ and ν follows [7], it is adapted and modified for our purpose. Let $(L, \leq)$ be a partial order. We try to get some information on the structure of L by considering certain “cardinal characteristics” $\mu(L)$ and $\nu(L)$, which are defined as follows: $\mu(L)$ is the smallest cardinal μ such that there is no uniform⁶ set $A \subseteq L$ of cardinality μ , $\mu_n(L) = \mu(L^n)$ for $n > 0$, $\nu(L_1, L_2)$ is the the smallest

⁶ either an antichain, a well-ordered chain, or a co-well-ordered chain.

cardinal ν such that there are no pairwise incomparable monotone functions $(f_i : i < \nu)$ from L_1 to L_2 , $\nu_n(L) = \nu(L^n, L)$, $\nu(L) = \nu_1(L)$, $\mu_\infty = \sup\{\mu_n : n \in \omega\}$ and $\nu_\infty = \sup\{\nu_n : n \in \omega\}$.

From the previous definitions, we trivially have that for all $n \in \omega$, $\mu_n \leq \mu_{n+1}$ and $\nu_n \leq \nu_{n+1}$.

Fact 1. *Let L be infinite. Then $\mu_n(L) \leq |L|^+$ and $\nu(L) \leq (2^{|L|})^+$.*

Our goal is to see the link with large cardinals, in particular strong inaccessible cardinals, and how those relations can be used to get a hint about $\mu(L)$ and $\nu(L)$. Partition relations (in the proof of the following proposition) help us in understanding μ and we will see later how this gives us a better understanding of ν .

Proposition 1. *Let $(L, \leq)$ be a partial order.*

- (a) $|L| \leq 2^{\mu(L)}$.
- (b) If κ is a strong limit cardinal, then $\kappa \leq \mu(L)$ iff $\kappa \leq |L|$.
- (c) If κ is a strong limit cardinal, then $|L| > \kappa$ implies $\mu(L) > \kappa$,
- (d) If κ is a strong limit cardinal, then $\mu(L) = \kappa$ implies $|L| = \kappa$.

And the following facts are well-known results, proved using an independence lemma of Shelah and Goldstern. It gives an evaluation (lower estimates) of ν for simple (uniform) sets that we use in order to evaluate μ , as soon as we have some strong relation between ν and μ .

- Fact 2.** (a) *If A is uniform, $|A| > 2$, then $\nu(A) > 2$.*
 (b) *If A is uniform, $|A| = \kappa \geq \aleph_0$, then $\nu(A) > 2^\kappa$, i.e., there are 2^κ many pairwise incomparable monotone functions from A to A .*
 (c) *If A is an antichain, $|A| = \kappa \geq \aleph_0$, then $2^\kappa < \nu(A, \{0, 1\})$, i.e., there are 2^κ many incomparable (necessarily monotone) functions from A into the two-element set $\{0, 1\}$.*

3 Relations between μ and ν

Recall that our final goal is to compare $\mathcal{C}$ and $\mathcal{C}' = \text{MONOTONE} - \text{FO}[\text{OPERATOR}^{(2)}]$ that verify the $(\star)$ conditions. We first want to show that if $\mathcal{C}$ and $\text{MONOTONE}[\text{OPERATOR}^{(2)}]$ correspond on structures of cardinality κ , then κ is a large cardinal.

We will now study the relation between μ and ν . To do that, we first look at μ_∞ and ν_∞ . Assuming that L is nice, we show in proposition 2 that the existence of many incomparable monotone functions from L^n to L implies the existence of a large uniform set in some L^m . Then we show in lemma 1 that a large uniform set in L^m implies the existence of many incomparable monotone functions from L^m to L . Finally, in theorem 3, we combine proposition 2 and lemma 1 to show that μ must be a strong limit cardinal.

First, we need to precise the underlying partial order in $\mathcal{A} = \langle 2^L, \preceq \rangle$. It is defined such that any monotone ^{$\preceq$} function can effectively be used with the OPERATOR^② operator of $\mathcal{C}$.

$$a \preceq b \quad \text{iff} \quad a \subseteq b \text{ and } a \leq b \setminus a$$

and a new incomparability notion (completely independent of the incomparability due to $\preceq$) :

$$a \mathbb{I} b \quad \text{iff} \quad \forall c \in a \forall d \in b \ c \parallel d$$

We now introduce $\nu'(L)$: it is the smallest cardinal ν' such that there is no family $(f_i : i < \nu')$ of pairwise incomparable ^{$\mathbb{I}$} monotone ^{$\preceq$} functions from $A = 2^L$ to A . In the following, when considering elements of A , *incomparable* stands for incomparable ^{$\mathbb{I}$} . Trivially, we have $\nu(L) \leq \nu'(L)$. We define also $\nu'_n(L) = \nu'(L^n)$ as we did for μ_n and ν_n . We could as well introduce $\nu''(L)$ as the smallest cardinal such that there is no family of pairwise incomparable monotone functions *definable in first order logic* but we would then get $\nu'' < \nu'$ which would not be of any help. So we decide to stick to monotone functions and we will end up with definable functions later on.

Proposition 2. *Let $(L, \leq)$ be a nice structure, κ a cardinal of cofinality $> cf(2^{\aleph_0})$. If $\kappa < \nu'_n(L)$, then $\kappa < \mu_\infty(L)$.*

Proof. Let us assume $\kappa < \nu'_n(L)$.

So, let $(f_i : i < \kappa)$ be a family of pairwise incomparable monotone functions from 2^{L^n} to 2^L . Since $\mathcal{L}$ is a nice structure, each of these functions with an OPERATOR^② operator can be written as t_i in $\mathcal{C}$. Thus, for each i there is some natural number k_i and a definable function $g_i(\bar{x}, y_1, \dots, y_{k_i})$ and a k_i -tuple $\bar{b}^i = (b_1^i, \dots, b_{k_i}^i)$ such that

$$t_i = [\text{OPERATOR}^{\textcircled{1}}g_i](\bar{b}^i)$$

Since there are only $\leq 2^{\aleph_0}$ many pairs (t_i, k_i) and we have assumed $cf(\kappa) > cf(2^{\aleph_0})$, we may assume that they are all equal, say to (t^*, k^*) . But then $(\bar{b}^i : i < \kappa)$ must be pairwise incomparable in $2^{L^{k^*}}$, because, with our assumptions on $\mathcal{C}$ and $\mathcal{C}'$, if $\bar{b}^i$ and $\bar{b}^j$ were comparable then f_i and f_j would be comparable. Hence we have found an antichain of size κ in $2^{L^{k^*}}$. And by definition of $\mathbb{I}$, this implies that we also have an antichain of size κ in L^{k^*} .

4 Main Result

To show the relative consistency of the infinite versions of $\mathcal{C} \neq \text{MONOTONE}[\text{OPERATOR}^{\textcircled{2}}]$, we first show (using all the previous lemmata) that the cardinality of L is necessarily a strong limit cardinal and then using a lemma of Goldstern and Shelah, that it is also regular.

In order to show the next important lemma, it is necessary to have a bounded (with a smallest and a greatest element) structure, which we have because of the implication of the $(\star)$ conditions on nice structures.

Lemma 1. *Let $(L, \leq, 0, 1)$ be a bounded partially ordered structure, κ an infinite cardinal. If $\kappa < \mu_n(L)$, then $2^\kappa < \nu_n(L)$. In particular, $\kappa < \mu_\infty$ implies $2^\kappa < \nu_\infty$.*

Theorem 3. *If L is infinite and nice, then*

- (a) $\mu_\infty(L)$ must be a strong limit cardinal,
- (b) $\mu(L) = \mu_\infty(L)$
- (c) $|L| = \mu(L)$.

Proof. (a) If $\kappa < \mu_\infty(L)$, then $2^\kappa < \nu_\infty(L)$ by lemma 1. So, $2^{2^\kappa} < \nu'_\infty(L)$. Now 2^{2^κ} always has cofinality greater than $2^{\aleph_0} \geq \text{cf}(2^{\aleph_0})$, so we get $2^{2^\kappa} < \mu_\infty(L)$ by proposition 2. And hence $2^\kappa < \mu_\infty(L)$.

(b) Assume that $\mu(L) < \mu_\infty(L)$. Let $\lambda = 2^{2^{\mu(L)}} < \mu_\infty(L)$. By proposition 1(a), $|L| \leq 2^{\mu(L)} < \lambda$, so $\mu_n(L) \leq |L|^+ \leq \lambda$ for all $n \in \omega$, hence $\mu_\infty(L) \leq \lambda$, a contradiction.

(c) Use proposition 1(d): Let $\kappa = \mu(L)$. From proposition 1(b) we conclude $\kappa \leq |L|$, and from proposition 1(c), we conclude $\kappa \geq |L|$.

We have shown that for a nice structure L the cardinal characteristic $\mu(L)$ must be a strong limit cardinal. Now we need to show that $\mu(L)$ must be regular. A lemma of Goldstern and Shelah [7, 4.1] shows that the singularity of $\mu(L)$ would imply the existence of many incomparable monotone functions (a lot more than $\mu(L)$), and we show that this would yield a contradiction.

Lemma 2. *Let $(L, \leq, 0, 1)$ be a bounded partially ordered structure, and let κ be a singular strong limit cardinal, $\kappa \leq |L|$.*

Then $\nu(L) > \kappa$.

If moreover $\text{cf}(\kappa) = \aleph_0$, then we even get $\nu(L) > 2^\kappa$.

Proof. See [7, 4.1].

Theorem 4. *If $(L, \leq)$ is a nice structure, then $\mu(L) = |L|$ is an inaccessible cardinal.*

Proof. Let $\kappa = \mu(L)$. From theorem 3, we know that κ is a strong limit, and that $|L| = \kappa$. Assume that κ is singular.

First, let us assume that $\text{cf}(\kappa)$ is uncountable. The previous lemma 2 tells us that $\nu(L) > \kappa$, so $\nu'(L) > 2^\kappa$. Now, we know that κ is a strong limit cardinal and so because of its singularity, $2^{\text{cf}(\kappa)} < \kappa$. Moreover, $\text{cf}(\kappa) > \aleph_0$, so $2^{\text{cf}(\kappa)} \geq 2^{\aleph_0} \geq \text{cf}(2^{\aleph_0})$ and by König's theorem, $\text{cf}(2^\kappa) > \kappa > \text{cf}(2^{\aleph_0})$. We can then apply our proposition 2 : $\mu_\infty(L) > 2^\kappa$, a contradiction.

Now we consider the second case: $\text{cf}(\kappa) = \aleph_0$. Here lemma 2 tells us $\nu(L) > 2^\kappa$ and so $\nu'(L) > 2^{2^\kappa}$. Since 2^{2^κ} has cofinality $> 2^{\aleph_0} \geq \text{cf}(2^{\aleph_0})$, we can again apply proposition 2 and again get $\mu_\infty(L) > 2^{2^\kappa} > \kappa$, a contradiction.

We then know that L has a strong inaccessible cardinality because $|L| = \mu(L)$ when L is infinite and nice.

Theorem 4 tells us that “ $\text{MONOTONE}[\text{OPERATOR}^{(2)}]$ is not equal to $\mathcal{C}$ on any infinite structures” is consistent relative to ZFC. Recall that our goal is to compare $\mathcal{C}$ to $\text{MONOTONE} - \text{FO}[\text{OPERATOR}^{(2)}]$. We are able to come to our ends through the following modified forcing theorem of Goldstern and Shelah [6]. It is easily shown that our partial order $\preceq$ (from $\mathcal{A}$) still verifies the forcing conditions of the theorem whenever we have an original partial order $\leq$ (from $\mathcal{L}$) that verifies them.

Theorem 5. *The statement*

“There is a partial order $(P, \preceq)$ such that all monotone $^{\preceq}$ functions $f : P \rightarrow P$ are definable in P ”

is consistent relative to ZFC. Moreover, the statement holds in any model obtained by adding (iteratively) ω_1 Cohen reals to a model of CH.

Hence we can reach our goal :

Theorem 6. “ $\text{MONOTONE} - \text{FO}[\text{OPERATOR}^{(2)}]$ is not equal to $\mathcal{C}$ on some infinite structure” is consistent relative to ZFC.

Proof. “(Strong) inaccessible cardinals do not exist” is consistent with the continuum hypothesis and also with Cohen reals. Therefore it is consistent with “ $\text{MONOTONE} = \text{MONOTONE} - \text{FO}$ on a particular structure”. And so we have, by the previous theorem and by theorem 4 that “on this particular structure, $\text{MONOTONE} - \text{FO}[\text{OPERATOR}^{(2)}] \neq \mathcal{C}$ ” is consistent with ZFC.

5 Concluding Remarks

One natural question is then to know if those questions, in the infinite case, are independent of the axioms of set theory. This does not appear to be an easy task.

One can also try to give some interpretation of our results in terms of special “infinite” Turing machines. Infinite time Turing machines have been considered in the past in various ways. Hamkins and Lewis [9] considered *countable* tape machines that at limit ordinal stages of the computation make their cell values the lim sup of the cell values (0 or 1) before the limit and enter a special distinguished limit state with the head of each tape plucked from wherever it might have been racing towards and placed on top of the first cell of that tape. At successor ordinal stages, they behave as classical Turing machines. Hamkins and Lewis also obtain some recursive analogues of large cardinals. For example, the supremum of the writable (countable ordinals are somehow coded in reals) ordinals is recursively inaccessible : it is recursively Π_1^1 -indescribable. So again, the large cardinals were predictable for special machines with *non-countable* tapes. With specific special infinite time and space Turing machines, one can give analogues of usual complexity class definitions using ordinal arithmetic (polynomial, logarithmic. . .) and even obtain the same logical characterizations as in the finite

Turing machine case. In this *infinite* Turing machine framework, our result then states that the separation of most complexity classes is relatively consistent with set theory for any infinite analogue of complexity class definitions, with the only requirement that our $(\star)$ conditions on the classes (finite and infinite versions) are met. What do the $(\star)$ conditions mean in this Turing context?

Another open question is whether there are some other complexity classes, apart from LOGSPACE, NLOGSPACE and P, that verify the $(\star)$ conditions and whose separations are not trivial in the infinite case.

Acknowledgments. The first author would like to thank his PhD advisors, Menachem Magidor and Jacques Mazoyer for their support, suggestions and remarks. We are also grateful to Martin Goldstern and Saharon Shelah for coming up with the ideas of their order polynomially complete lattices paper [7] and thus showing us the path to follow.

References

1. S. Abiteboul, M.Y. Vardi, and V. Vianu, *Fixpoint logics, relational machines, and computational complexity*, Proceedings of the 7th IEEE Symposium on Logic in Computer Science, 1992, pp. 156–168.
2. H.-D. Ebbinghaus and J. Flum, *Finite model theory*, Springer-Verlag, Berlin, 1995.
3. P. Erdős, A. Hajnal, A. Máté, and R. Rado, *Combinatorial set theory : Partition relations for cardinals*, North-Holland, Amsterdam, 1975.
4. L. Fortnow, S. Kurtz, and D. Whang, *The infinite version of an open communication complexity problem is independent of the axioms of set theory*, SIGACT News **25** (1994), no. 1, 87–89.
5. M. Furst, J. Saxe, and M. Sipser, *Parity, circuits and the polynomial time hierarchy*, Mathematical Systems Theory **17** (1984), 13–27.
6. M. Goldstern and S. Shelah, *A partial order where all monotone maps are definable*, Fundamenta Mathematicae **152** (1997), 255–265.
7. ———, *Order polynomially complete lattices must be large*, Algebra Universalis (1998), to appear.
8. Y. Gurevich and S. Shelah, *Fixed-point extensions of first-order logic*, Annals of Pure and Applied Logic **32** (1986), 265–280.
9. J. D. Hamkins and A. Lewis, *Infinite time turing machines*, preprint, June 1997.
10. T. Jech, *Set theory*, Academic Press, New York, 1978.
11. A. Kanamori, *The higher infinite*, Springer Verlag, 1994.
12. A. Kanamori and M. Magidor, *The evolution of large cardinal axioms in set theory*, Higher Set Theory (Gert H. Muller and Dana S. Scott, eds.), Lecture Notes in Mathematics, vol. 669, Springer Verlag, Berlin, 1978, pp. 99–275.
13. A. Miller, *On the length of Borel hierarchies*, Annals of Mathematical Logic **16** (1979), 233–267.
14. W. Richter, *Recursively mahlo ordinals and inductive definitions*, Logic Colloquium '69 (R. O. Gandy and C. E. M. Yates, eds.), North-Holland, 1971, pp. 273–288.
15. M. Sipser, *Borel sets and circuit complexity*, Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 1983, pp. 61–69.

Timed Automata with Monotonic Activities ^{*}

Ruggero Lanotte and Andrea Maggiolo-Schettini

Dipartimento di Informatica, Università di Pisa, Corso Italia 40,
56125 Pisa, Italy
{lanotte, maggiolo}@di.unipi.it

Abstract. The paper introduces *TAMA* (Timed Automata with Monotonic Activities) a subclass of hybrid automata including as proper subclass *TA* (Timed Automata), Multirate Automata and Integrator Automata. A subclass of *TAMA*, called *TAMA_o*, is studied and shown to be equivalent to Timed Automata under the assumption of discrete time, while in dense time *TAMA_o* contains properly the subclasses of hybrid systems mentioned. We also show that *TAMA_o* allow more succinct descriptions than *TA*.

1 Introduction

In recent years a number of automata have been proposed for modeling real-time systems. The behaviour of such systems is described in terms of acceptance or non acceptance of timed sequences (namely of sequences of symbols, or sets of symbols, annotated with a time value)(see [1], [2] and [7]). Automata such as the ones mentioned are finite automata equipped with a set of variables. With the states of an automaton an evolution law is associated which gives the value of variables with time passing. Transitions of the automaton are labeled with guarded sets of assignments to variables. The most general case is that of hybrid automata ([1]). Among their subclasses with linearly changing variables we have Multirate Automata ([1]), Integrator Automata ([1]) and Timed Automata ([2]). Timed Automata (*TA*) have decidable properties both with discrete and dense time domain assumption. Now, for the purpose of modelling real systems the assumption that variables change linearly seems to be restrictive. In this paper we introduce Timed Automata with Monotonic activities (*TAMA*), a subclass of hybrid automata which is a superclass of the subclasses mentioned above, with sequences of timed sets of symbols as input, and limited to finite sequences. In our model each state is labeled with monotonic functions (giving for each variable its evolution law), and each transition is labelled with a set of symbols, a condition and a subset of variables which are reset. We consider a subclass of *TAMA*, called *TAMA_o*, characterized by limitations on how the evolution law of a specific variable may vary when passing from a state to another. In the dense time case the subclass *TAMA_o* contains the subclasses of hybrid automata mentioned above, and therefore undecidability results proven for these classes hold also for

^{*} Research partially supported by MURST Progetto Cofinanziato TOSCA.

$TAMA_o$. In the discrete time case $TAMA_o$ are shown to be equivalent to TA , and therefore they enjoy the same properties of discrete TA . Finally we show that both in the discrete and in the dense time domain $TAMA_o$ allow more succinct descriptions than TA . We have arguments to conjecture that we may express properties that when hold in the discrete time domain hold also in the dense time domain.

2 Definitions

We define sets of unary functions from rationals to rationals:

$$F_+ = \{f : \mathbb{Q} \rightarrow \mathbb{Q} \mid x_1 \leq x_2 \text{ then } f(x_1) \leq f(x_2) \text{ with } f(0) = 0\},$$

the set of increasing monotonic functions which assume value 0 for abscissa 0,

$$F_- = \{f : \mathbb{Q} \rightarrow \mathbb{Q} \mid x_1 \leq x_2 \text{ then } f(x_1) \geq f(x_2) \text{ with } f(0) = 0\},$$

the set of decreasing monotonic functions which assume value 0 for abscissa 0,

$$F_\infty = \left\{f : \mathbb{Q} \rightarrow \mathbb{Q} \mid \lim_{x \rightarrow \infty} f(x) = \pm\infty\right\},$$

the set of functions which tend to infinite for value of abscissa tending to infinite,

$$F_s = \{f : \mathbb{Q} \rightarrow \mathbb{Q} \mid \exists n, h \in \mathbb{Q} \forall h' > h \ f(h') = n\},$$

the set of functions which, beginning from a certain value, assume a constant value. If f is a function and c is a constant then we denote with $c_1 f + c_2$ the function such that for all x $(c_1 f + c_2)(x) = c_1 f(x) + c_2$. We assume a set Φ of conditions over the variables, defined as follows:

$$\phi ::= true \mid x = c \mid x > c \mid \phi_1 \wedge \phi_2 \mid \neg \phi_1,$$

with $x \in X$ and $c \in \mathbb{Q}$. For X a set of variables, an *evaluation* v is a function $v : X \rightarrow \mathbb{Q}$. We say $v \models \phi$ if and only if ϕ is true for the value of variables given by v .

Let $\mathbb{N}_k = \{kn \mid n \in \mathbb{N}\}$ with $k \in \mathbb{Q}$. By *Time* we may mean either the dense time $\mathbb{Q}$ or the discrete time $\mathbb{N}_k$ (it will be clear from the context what is the case). We call k *time step*.

We define a *Timed Automaton with Monotonic Activities (TAMA)* an automaton with variables assuming values given by a monotonic function over *Time*. A specific monotonic function is associated with each variable at any state by an activity function. A transition from state to state depends on a set of symbols being read and a condition on values of variables, and, when performed, resets a subset of variables.

A *TAMA* is a tuple $T = \langle Q, q_0, \Sigma, X, Act, E, F \rangle$ with

- Q finite set of states

- $q_0 \in Q$ initial state
- Σ finite input alphabet
- X finite set of variables
- $Act : (Q \times X) \rightarrow (F_+ \cup F_-)$ activity function
- $E \subseteq Q \times 2^\Sigma \times \Phi \times 2^X \times Q$ finite set of transitions
- $Q_f \subseteq Q$ set of final states.

In a state q a variable x varies with time, assuming values given by $Act(q, x) + c$ for some constant c .

Example 1. Assume we have two basins, and basin 1 communicates with basin 2. Basin 1 has two taps one to fill the basin and the other to open the communication with basin 2. Basin 2 has a tap to empty the basin. Tap 2 cannot be opened concurrently with tap 1 or tap 3 (see Fig. 1). Assume also that the basins can be filled by a liquid up to height 25 and that, once taps are open, the level of liquid varies as a quadratic function of time. Finally assume that once a tap is opened to empty a basin the whole basin is emptied. We denote with

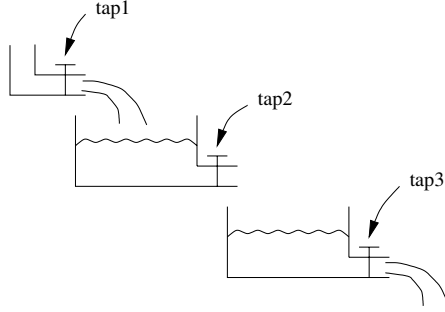


Fig. 1. Communicating basins

f, f' the functions such that for each $t \in \mathbb{Q}$

$$f(t) = \begin{cases} t^2 & \text{if } t \leq 5 \\ 25 & \text{otherwise} \end{cases}$$

and $f'(t) = 0$. We denote with $-f$ the function $(-1)f$. We use the variables x, y to denote the level of basin 1 and basin 2, respectively. Input symbols o_1, o_2, o_3 stand for open tap 1, open tap 2, open tap 3, respectively. Input symbols c_1, c_2, c_3 stand for close tap 1, close tap 2, close tap 3, respectively. The activities of the variables x and y are represented by a pair where the first element is the activity of x and the second one is the activity of y . The basins are modelled by the automaton of Fig. 2.

With C_x^{max} (resp. C_x^{min}) we denote the max (resp. min) constant with which the variable x is compared in constraints of transitions. In the case of the example we have $C_x^{max} = 25$ and $C_x^{min} = 0$.

A *timed word* of length k is a pair $\langle \alpha_1, \alpha_2 \rangle$ with $\alpha_1 : [0, k-1] \rightarrow 2^\Sigma$ and $\alpha_2 : [0, k-1] \rightarrow Time$ such that $\alpha_2(i) \leq \alpha_2(i+1)$.

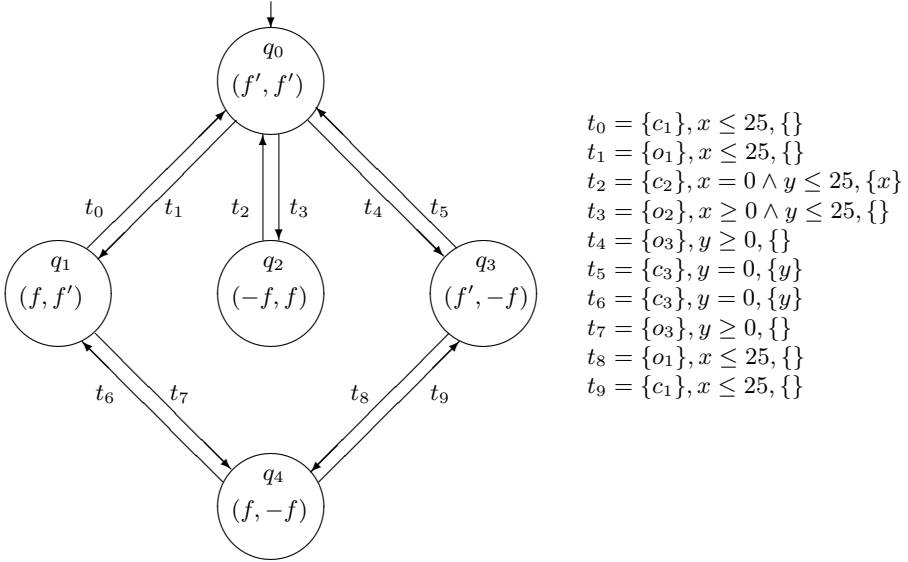


Fig. 2. An automaton with monotonic activities

A *configuration* is a triple $\langle q, \Psi, \tau \rangle$ where

1. q is the current state
2. $\Psi : X \rightarrow (F_+ \cup F_-)$ is a function associating a function with each variable q , which says how values of variables vary with time in state q
3. $\tau : X \rightarrow \text{Time}$ is a function associating with each variable x the amount of time for which x has been varying with the function $\Psi(x)$.

The functions Ψ and τ define an evaluation $v_\Psi^\tau : X \rightarrow \mathbb{Q}$ such that $v_\Psi^\tau(x) = \Psi(x)(\tau(x))$.

A *step* is

$$\langle q_1, \Psi_1, \tau_1 \rangle \xrightarrow{t}_A \langle q_2, \Psi_2, \tau_2 \rangle$$

where

$$- \langle q_1, A, \phi, Y, q_2 \rangle \in E$$

$$- \Psi_2(x) = \begin{cases} \Psi_1(x) & \text{if } x \notin Y \wedge \text{Act}(q_1, x) = \text{Act}(q_2, x) \\ \text{Act}(q_2, x) & \text{if } x \in Y \\ \text{Act}(q_2, x) + \Psi_1(x)(\tau_1(x) + t) & \text{otherwise} \end{cases}$$

$$- \tau_2(x) = \begin{cases} \tau_1(x) + t & \text{if } x \notin Y \wedge \text{Act}(q_1, x) = \text{Act}(q_2, x) \\ 0 & \text{otherwise} \end{cases}$$

$$- v_\Psi^{\tau_1+t} \models \phi.$$

In the new configuration if the variable x has not been reset and q_1 and q_2 have the same activity then $\Psi_1(x) = \Psi_2(x)$ and the amount of time for which x has been varying with the function $\Psi_1(x)$ is increased by t . Such amount of time is zero otherwise. This means that when in the new state the activity function for a variable x does not vary, x assumes the successive values of $\Psi_1(x)$. Note that this is not the case in hybrid automata (see [1]) where at each change of state, also if the function for x does not vary, the values of x are computed starting anew from zero. Our choice is motivated by the need of describing transitions which do not alter the behaviour of certain variables. Note that, anyway, the choice of hybrid automata can be simulated in our automata, and viceversa.

Example 2. Let us consider the automaton of Fig.2. We have the step

$$\langle q_1, (f, f'), (0, 2) \rangle \xrightarrow{4}_{\{0_3\}} \langle q_4, (f, -f), (4, 0) \rangle.$$

We have that $v_{(f, -f)}^{(4, 0)}(x) = f(4) = 16$ and $v_{(f, -f)}^{(4, 0)}(y) = -f(0) = 0$.

The value of x in the hybrid case and in the TAMA case are described in Fig. 3.

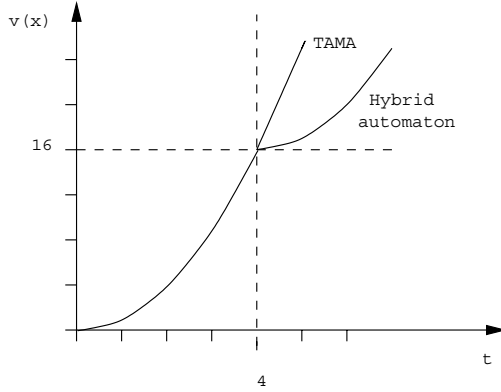


Fig. 3. Evolution of x

A *run* over a timed word $\langle \alpha_1, \alpha_2 \rangle$ of length n is a sequence of steps

$$\langle q_0, \Psi_0, \tau_0 \rangle \xrightarrow{t_0}_{\alpha_1(0)} \langle q_1, \Psi_1, \tau_1 \rangle \dots \langle q_{n-1}, \Psi_{n-1}, \tau_{n-1} \rangle \xrightarrow{t_{n-1}}_{\alpha_1(n-1)} \langle q_n, \Psi_n, \tau_n \rangle$$

where q_0 is the initial state and, for all x , $\tau_0(x) = 0$ and $\Psi_0(x) = Act(q_0, x)$.

A run is *accepted* if $q_n \in Q_f$, $\mathcal{L}(T)$ is the set of timed words accepted by T with $Time = \mathbb{Q}$, $\mathcal{L}_k(T)$ is the set of timed words accepted by T with $Time = \mathbb{N}_k$.

Example 3. The language recognized by the automaton of Fig. 2 with $Q_f = \{q_4\}$ and dense time assumption is the set of pairs $\langle \alpha_1, \alpha_2 \rangle$ such that, for each j , if $\alpha_1(j) = \{o_2\}$ then $\alpha_1(j+1) = \{c_2\}$, and if $\alpha_1(j) = \{o_i\}$ with $i \neq 2$ then there

exists $k > j$ such that $\alpha_1(k) = \{c_i\}$ and between j and k there is not a symbol in $\{o_2, c_2\}$. If s_i^k is the sum of $(\alpha_2(h) - \alpha_2(h'))^2$ such that $k \geq h > h'$ and h' is minimum such that $\alpha_1(h) = \{c_i\}, \alpha_1(h') = \{o_i\}$ then, for each k and $i = 1, 2$, we have $s_{i+1}^k = s_i^k \leq 25$.

Let $\mathcal{L}_k(T)h = \{\langle \alpha_1, \alpha_2 \rangle \mid \langle \alpha_1, \alpha'_2 \rangle \in \mathcal{L}_k(T) \wedge \alpha_2(i) = h\alpha'_2(i)\}$. The product of an automaton T with a constant h is an automaton hT where all the activities Act' of hT are translated by h w.r.t. the activities Act of T , namely $Act'(q, x)(t) = Act(q, x)(ht)$.

Proposition 1. $\mathcal{L}_{kh}(hT) = \mathcal{L}_k(T)h$ for each $h, k \in \mathbb{Q}$.

By Proposition 1, it will be enough for our purposes to assume time step 1.

3 Expressiveness and Closure Results

Given a *TAMA* T we define a relation on the activities of a variable x :

$Act(q_1, x) \rightarrow_x Act(q_2, x)$ iff there exist $\langle q_1, A, \phi, Y, q_2 \rangle$ such that $x \notin Y$.

The relation $\rightarrow_x$ expresses that the last value of x in q_1 according to $Act(q_1, x)$ is equal to the first value of x in q_2 and, since then, x varies according to $Act(q_2, x)$. With $\rightarrow_x^*$ we denote the transitive closure of $\rightarrow_x$ and with f^0 we denote the function that assigns 0 for each time t , namely $f^0(t) = 0$.

We consider a class of automata called *TAMA_o* which are characterized by the fact that activities are monotonic functions in $F_\infty \cup F_s$ and have the property that the function associated with a variable may change from increasing to decreasing (or viceversa) and, after changing, values are given by a function in F_s , namely $T \in TAMA_o$ iff

1. for any variable x and state q of T we have $Act(q, x) \in (F_\infty \cup F_s)$
2. if x is a variable of T and F_x is the set of activities associated with x in T , for $f_1, f_2 \in (F_x \cap F_+ - \{f^0\})$ and $f_3, f_4 \in (F_x \cap F_- - \{f^0\})$ we have

if $f_1 \rightarrow_x^* f_3$ and $f_1 \rightarrow_x^* f_4$ then $f_3 = f_4, f_3 \in F_s$ and $f_3 \not\rightarrow_x^* f_1$

or

if $f_3 \rightarrow_x^* f_1$ and $f_3 \rightarrow_x^* f_2$ then $f_1 = f_2, f_1 \in F_s$ and $f_1 \not\rightarrow_x^* f_3$.

Example 4. The automaton of Fig 2 is a *TAMA_o*.

Timed Automata TA on finite sequences of sets of symbols are obtained by assuming that, for any state q and variable x , $Act(q, x)(t) = t$. In the definition of step of *TAMA* we require equality between input symbols at an instant and set of symbols in the label of the transition which may be triggered, and therefore these sets of symbols may be codified with a unique symbol. Such *TA* on finite sequences of symbols enjoy the same properties of *TA* defined on infinite words as in [2] (see [3]). In dense time domain is immediate to prove that the class of languages recognized by *TAMA_o* strictly contains the class of languages recognized by *TA* (as Multirate Automata, see [1], are a subclass of *TAMA_o*).

Proposition 2. $\mathcal{L}(TA) \subset \mathcal{L}(TAMA_o)$.

This is not true in the discrete time domain. Actually we prove that in discrete time, for each $k \in \mathbb{Q}$, we have $\mathcal{L}_k(TA) = \mathcal{L}_k(TAMA_o)$.

Without loss of generality (see Proposition 1), we consider the case with time step equal to 1. We prove that the values of variables that we consider for simulation are finitely many.

For $T \in TAMA_o$ we define the constant t_f^x , for f an activity of a variable x :

1. if $f \in F_+ \cap F_\infty$ then

$$t_f^x = \min\{h \in \mathbb{N} \mid \forall h' > h, \forall \text{Act}(q, x) \in F_s \text{ Act}(q, x)(h) + f(h') > C_x^{max}\}$$

namely t_f^x gives the minimum time (over the naturals) such that for times higher than t_f^x the value of x is always greater than C_x^{max} (note that such minimum always exists by definition of F_s and F_∞).

2. if $f \in F_- \cap F_\infty$ then

$$t_f^x = \min\{h \in \mathbb{N} \mid \forall h' > h \forall \text{Act}(q, x) \in F_s \text{ Act}(q, x)(h) + f(h') < C_x^{min}\}$$

namely t_f^x gives the minimum time (over the naturals) such that for times higher than t_f^x the value of x is always lower than C_x^{min} (note that such minimum always exists by definition of F_s and F_∞).

3. if $f \in F_s$ then $t_f^x = \min\{h \in \mathbb{N} \mid \exists n \forall k > h f(k) = n\}$ namely t_f^x gives the time (over the naturals) such that for times higher than t_f^x the value of x is a constant.

With t_x we denote the maximum constant t_f^x and with $\hat{t}$ the maximum constant t_x , namely $t_x = \max\{t_{\text{Act}(q,x)}^x \mid q \in Q\}$ and $\hat{t} = \max\{t_x \mid x \in X\}$.

The following lemma shows that the values assumed by a variable x in the interval $[C_x^{min}, C_x^{max}]$ are finitely many.

Lemma 1. *With discrete time domain assumption, for each variable x of an automaton $T \in TAMA_o$, there exists a finite set $I \subseteq \mathbb{Q}$ such that for each run*

$$\langle q_0, \Psi_0, \tau_0 \rangle \rightarrow_{\alpha_1(0)}^{t_0} \langle q_1, \Psi_1, \tau_1 \rangle \rightarrow_{\alpha_1(1)}^{t_1} \dots \langle q_{n-1}, \Psi_{n-1}, \tau_{n-1} \rangle \rightarrow_{\alpha_1(n-1)}^{t_{n-1}} \langle q_n, \Psi_n, \tau_n \rangle$$

$$\text{if } v_{\Psi_h}^{\tau_h + t_h}(x) \in [C_x^{min}, C_x^{max}] \text{ then } v_{\Psi_h}^{\tau_h + t_h}(x) \in I.$$

Proof. (Idea) Each evaluation $v_{\Psi_h}^{\tau_h + t_h}$ such that $v_{\Psi_h}^{\tau_h + t_h}(x) \in [C_x^{min}, C_x^{max}]$ can be proved to be a linear combination of finite values and finite indices. If f is an activity of x , we define the natural constant s_f equal to the maximum number of steps $|f(t) - f(t-1)| \neq 0$ with $t \in [1, \hat{t} + 1]$, when one assumes x given by f and the value of x in the interval $[C_x^{min}, C_x^{max}]$. Therefore by definition of s_f and t_f^x if $v_{\Psi_h}^{\tau_h + t_h}(x) \in [C_x^{min}, C_x^{max}]$ then $v_{\Psi_h}^{\tau_h + t_h}(x) = \sum_{f \in \{\text{Act}(q,x) \mid q \in Q\}} \sum_{c_f \in D_f} i_f c_f$

with $i_f \in [0, s_f]$ (number of steps) and where $D_f = \{f(t) \mid t \in [0, t_f^x]\}$ (possible function values). $\square$

Theorem 1. (*Equivalence*) For any $TAMA_o$ T there exists a TA T' such that $\mathcal{L}_1(T) = \mathcal{L}_1(T')$.

Proof. (Idea) For each variable x we denote with I_x the finite set of values $I \cup \{Act(q, x)(t_x) | q \in Q\}$, where I is the set of Lemma 1 and with $[h]$ we denote the set $\{0, \dots, h\}$. We can store in a state of T' the information about the time for which the function given by the current activity has been applied, and the starting value of the function for each variable. Moreover, by definition of t_x , each time greater than t_x may be reduced to t_x , and therefore the number of constants we have to consider is finite, and by Lemma 1 also starting values are in a finite number.

Let $T = \langle Q, q_0, \Sigma, X, Act, E, Q_f \rangle$ with $X = \{x^0, \dots, x^m\}$. We can define $T' = \langle Q', q'_0, \Sigma, \{x\}, Act', E', Q'_f \rangle$ such that $Q' = Q \times I_{x^0} \times \dots \times I_{x^m} \times [t_{x^0}] \times \dots \times [t_{x^m}]$, $q'_0 = [q_0, 0, \dots, 0]$, $Act'(x)(t) = t$ and E' is a finite set of transitions $\langle [q, i_0, \dots, i_m, t_0, \dots, t_m], A, \phi', [q', i'_0, \dots, i'_m, t'_0, \dots, t'_m] \rangle$, where ϕ' may be either $x = c$ with $c \in [0, \dots, \hat{t} - 1]$ or $x \geq \hat{t}$ iff in T there exists a transition $\langle q, A, \phi, Y, q' \rangle$ that from the state $[q, i_0, \dots, i_m, t_0, \dots, t_m]$ takes to the state in $[q', i'_0, \dots, i'_m, t'_0, \dots, t'_m]$ in a time expressed by condition ϕ' . The set of final states Q'_f is $Q_f \times I_{x^0} \times \dots \times I_{x^m} \times [t_{x^0}] \times \dots \times [t_{x^m}]$. $\square$

Corollary 1. For any k and $T \in TAMA_o$ an automaton $T' \in TA$ exists such that $\mathcal{L}_k(T) = \mathcal{L}_k(T')$.

For the class TA it is known that universality, emptiness and reachability problem in the discrete time domain are decidable and, for Corollary 1, this holds also for the class $TAMA_o$. In the dense time domain these problems are undecidable for the class $TAMA_o$ (Integrator Automata are a subclass of $TAMA_o$).

The following lemmas and their corollary establish relationships between the language of a $TAMA_o$ T under dense time assumption and the language of T under assumption of discrete time domain.

Lemma 2. For any k and $T \in TAMA_o$ we have $\mathcal{L}_k(T) \subseteq \mathcal{L}(T)$.

Lemma 3. For any $T \in TAMA_o$ if $\alpha \in \mathcal{L}(T)$ then there exists k such that $\alpha \in \mathcal{L}_k(T)$.

Corollary 2. For any $T \in TAMA_o$ we have:

1. for any k if $\mathcal{L}_k(T) \neq \emptyset$ then $\mathcal{L}(T) \neq \emptyset$
2. there exists k such that if $\mathcal{L}(T) \neq \emptyset$ then $\mathcal{L}_k(T) \neq \emptyset$.

Let us recall some results we have for TA . In the dense time domain we have closure for union, intersection but not for complement. In the discrete time domain we have closure also for complement. This follows from the fact that, in the discrete time domain, for each non deterministic automaton a deterministic one exists that accepts the same language. Without loss of generality, we consider

the TA with only one clock (see [6]). Suppose that the time step (of discrete domain) is 1. The proof of equivalence is based on the following idea. Differently w.r.t. classical finite automata, transitions may not only depend on the same symbol but also reset the unique clock x (conditions do not pose problems). To simulate this clock in the deterministic automaton we take C_x^{max} clocks. In the deterministic automaton states are subsets of pairs of states (of the non deterministic automaton) and clocks. In the pair (q_i, x_i) , x_i is the current clock of q_i . Conditions associated with transitions exiting from q_i must be evaluated over x_i . Assume that in the non deterministic automaton for an input symbol a from a state q we have two transitions taking to states q' and q'' , respectively, and such that one resets the clock x and the other does not. In the deterministic automaton we shall have one only transition resetting a clock different from the current one of q , say x_i , and taking to a state $\{(q', x_j), (q'', x_i)\}$ where x_j is the new clock. We are ensured that C_x^{max} clocks are sufficient because after considering C_x^{max} resettings of clocks, a clock which has not been reset can be used as a new clock. So we have immediately the following proposition.

Proposition 3. *For each k $\mathcal{L}_k(TAMA)_o$ is closed under union, intersection and complement.*

4 Succinctness

Let us assume as a size of an automaton (of any of the two classes, TA and $TAMA_o$ considered) the sum of number of states, number of variables and number of conjunctions in conditions.

Given two classes of automata $\mathcal{A}$ and $\mathcal{B}$, the class $\mathcal{B}$ is *more succinct* than the class $\mathcal{A}$ (see [4]) if

1. for each automaton $A \in \mathcal{A}$ accepting the language $\mathcal{L}$ there exists an automaton $B \in \mathcal{B}$ accepting $\mathcal{L}$ and such that size of B is polynomial in the size of A
2. there is a family of languages L_n , for $n > 0$, such that each L_n is accepted by the automaton $B \in \mathcal{B}$ of a size polynomial in n , but the smallest $A \in \mathcal{A}$ accepting L_n is at least of size exponential in n .

Theorem 2. *With discrete time domain assumption, the class $TAMA_o$ is more succinct than the class TA .*

Proof. (Idea) We consider a language L_n of timed words $\langle \alpha_1, \alpha_2 \rangle$ with $\alpha_1 \in (\{0\} + \{1\})^* \{\text{stop}\}$ and α_2 such that

$$\sum_{i \text{ s.t. } \alpha_1(i)=\{0\}} \alpha_2(i+1) - \alpha_2(i) = \sum_{i \text{ s.t. } \alpha_1(i)=\{1\}} \alpha_2(i+1) - \alpha_2(i) = n.$$

It is easy to give a $TAMA_o$ with two variables working as stopwatches such that one advances while the other does not vary, depending on the symbol read. A TA recognizing the same language must consider all the possible combinations. $\square$

Theorem 3. *With dense time domain assumption, the class $TAMA_o$ is more succinct than the class TA .*

Proof. (Idea) We consider a language L_n over the alphabet $\Sigma = \{0, 1\}$ and such that $\{\langle \alpha_1, \alpha_2 \rangle \mid \alpha_1(i) = \alpha_1(i+n) \wedge \alpha_2(i) < \alpha_2(i+1), 0 \leq i \leq n-1\}$. It is easy to given a $TAMA_o$ with $2n$ stopwatches $x_{i,b}$ which are increased at iteration $i \in [0, n-1]$ if the symbol read is b . At iteration $i+n$ if the symbol read is b one must check whether $x_{i,b} > 0$. A TA recognizing the same language must consider all the possible cases. $\square$

5 Future Work

By Lemma 2, Lemma 3 and Corollary 2 we conjecture that, along the lines of [5], we could define a suitable decidable logic in which we may express properties that when hold in the discrete time domain (for a certain time step) hold also in the dense time domain. We conjecture also that both in dense and in discrete time domain the class of languages recognized by $TAMA_o$ is strictly included in the class of languages recognized by $TAMA$. If our conjecture is valid we would have the following inclusions

$$\mathcal{L}(TA) \subset \mathcal{L}(TAMA_o) \subset \mathcal{L}(TAMA)$$

and for any $k \in \mathbb{Q}$

$$\mathcal{L}_k(TA) = \mathcal{L}_k(TAMA_o) \subset \mathcal{L}_k(TAMA).$$

References

1. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J. and Yovine, S.: The Algorithmic Analysis of Hybrid Systems, Theoretical Computer Science **138** (1995) 3–34
2. Alur, R., Dill, D.: A Theory of Timed Automata, Theoretical Computer Science **126**, (1994) 183–235
3. Alur, R., Limor, F., Henzinger, T. A.: Event-clock automata: a determinizable class of timed automata, Theoretical Computer Science **211** (1999) 253–273
4. Drusinsky, D., Harel, D.: On the Power of Bounded Concurrency I: Finite Automata, Journal of ACM **41** (1994) 517–539
5. Henzinger, T. A., Manna, Z., Pnueli, A.: What Good are Digital Clocks? In: Kuich, W. (ed.): Automata, languages, and Programming. Lecture Notes in Computer Science, Vol. 623, Springer-Verlag, Berlin Heidelberg New York (1992) 545–558
6. Henzinger, T. A., Kopke, P.W., Wong-Toi, H.: The Expressive Power of Clocks. In: Fülöp, Z., Géseg, F. (eds.): Automata, languages, and Programming. Lecture Notes in Computer Science, Vol. 944, Springer-Verlag, Berlin Heidelberg New York (1995) 335–346
7. Lanotte, R., Maggiolo-Schettini, A., Peron, A.: Timed Cooperating Automata, Fundamenta Informaticae **42** (2000) 1–21

On a Generalization of Bi-Complement Reducible Graphs *

Vadim V. Lozin

University of Nizhny Novgorod, Gagarina 23, Nizhny Novgorod, 603600 Russia
lozin@unn.ac.ru

Abstract. A graph is called complement reducible (a *cograph* for short) if every its induced subgraph with at least two vertices is either disconnected or the complement to a disconnected graph. The bipartite analog of cographs, bi-complement reducible graphs, has been characterized recently by three forbidden induced subgraphs: $Star_{1,2,3}$, Sun_4 and P_7 , where $Star_{1,2,3}$ is the graph with vertices a, b, c, d, e, f, g and edges $(a, b), (b, c), (c, d), (d, e), (e, f), (d, g)$, and Sun_4 is the graph with vertices a, b, c, d, e, f, g, h and edges $(a, b), (b, c), (c, d), (d, a), (a, e), (b, f), (c, g), (d, h)$. In the present paper, we propose a structural characterization for the class of bipartite graphs containing no graphs $Star_{1,2,3}$ and Sun_4 as induced subgraphs. Based on the proposed characterization we prove that the clique-width of these graphs is at most five that leads to polynomial algorithms for a number of problems which are NP-complete in general bipartite graphs.

1 Introduction

The class of complement reducible graphs (*cographs* for short) was rediscovered independently by many researchers under different names and has been studied extensively over the years because of remarkable properties of these graphs. By definition, a graph G is a cograph if for any induced subgraph H of G with at least two vertices, either H or the complement to H is disconnected. The property of decomposability provides for cographs polynomial-time algorithms to solve many problems which are NP-complete in general graphs. In addition, cographs have a nice characterization in terms of forbidden induced subgraphs: these are exactly P_4 -free graphs, i.e. graphs containing no chordless path on 4 vertices P_4 as an induced subgraph [1]. It is no wonder that the results obtained for cographs have motivated researchers to investigate some generalizations of cographs, like P_4 -reducible graphs [8], P_4 -sparse graphs [9], semi- P_4 -sparse graphs [5], $(P_5, \overline{P_5})$ -free graphs [4], tree-cographs [10]. Moreover, the bipartite analog of cographs,

* This research has been supported by the Russian Foundation for Basic Research (Grant 00-01-00601). Part of the study has been done while the author was visiting RUTCOR, Rutgers Center for Operations Research, Rutgers University. The support of the Office of Naval Research (Grant N00014-92-J-1375) and the National Science Foundation (Grant DMS-9806389) is gratefully acknowledged.

bi-complement reducible graphs (short *bi-cographs*), has been introduced recently in [7] and has been characterized there by three forbidden induced subgraphs: $Star_{1,2,3}$, Sun_4 and P_7 , where $Star_{1,2,3}$ and Sun_4 are graphs depicted in Fig. 1, and P_7 is the chordless path on 7 vertices.

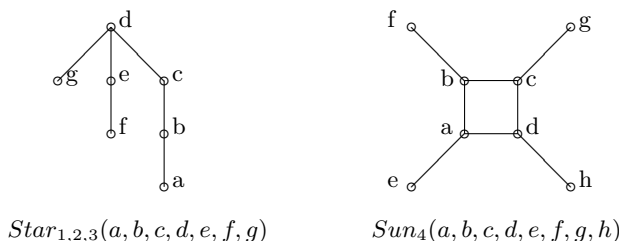


Fig. 1. Forbidden graphs

In the present paper we study a generalization of bi-cographs defined by two forbidden induced subgraphs: $Star_{1,2,3}$ and Sun_4 (Fig. 1). We provide the class of $(Star_{1,2,3}, Sun_4)$ -free bipartite graphs with a structural characterization, and then deduce from it that the clique-width of these graphs is at most five. The latter fact in combination with the results of [3] leads to polynomial time algorithms for a number of problems which are NP-complete for general bipartite graphs.

All graphs considered are undirected, without loops and multiple edges. A bipartite graph $H = (W, B, E)$ consists of set W of white vertices and set B of black vertices and set of edges $E \subseteq W \times B$. Sets W and B are called parts of H . For a bipartite graph $H = (W, B, E)$, we denote by $\tilde{H}$ the bipartite complement of H , i.e., $\tilde{H} = (W, B, (W \times B) - E)$.

The set of vertices and the set of edges of a graph G are denoted VG and EG , respectively. Also, $N(x) = \{y : (x, y) \in EG\}$ is the neighbourhood of a vertex $x \in VG$, and $\tilde{N}(x)$ is the neighbourhood of x in the bipartite complement to G . For a subset of vertices $U \subseteq VG$, we denote $N_U(x) = N(x) \cap U$, and $\tilde{N}_U(x) = \tilde{N}(x) \cap U$, and $G[U]$, the subgraph of G induced by set U . As usual, K_n , P_n , C_n , and $K_{n,m}$ denote, respectively, the complete graph, the chordless path, the chordless cycle with n vertices, and the complete bipartite graph with parts of cardinality n and m . By $2K_2$ we denote the disjoint union of two copies of K_2 .

A bipartite graph will be called *prime* if any two distinct vertices of the graph have different neighbourhoods.

2 Characterization of $Star_{1,2,3}$ -Free and Sun_4 -Free Bipartite Graphs

Theorem 1. *Let $G = (W, B, E)$ be a prime bipartite $(Star_{1,2,3}, Sun_4)$ -free graph. If G is connected and $\tilde{G}$ is connected, then either G or $\tilde{G}$ is $K_{1,3}$ -free.*

Proof. If G does not contain an induced P_7 , then G is a bi-cograph due to the result of [7]. This means that if G is connected and $\tilde{G}$ is connected, then G is a single-vertex graph and hence is $K_{1,3}$ -free.

Suppose now, without loss of generality, that set $U = \{1, 2, 3, 4, 5, 6, 7\}$ induces $P_7 = (1, 2, 3, 4, 5, 6, 7)$ in G with $1, 3, 5, 7 \in W$ and $2, 4, 6 \in B$. It is not hard to verify that set U induces in $\tilde{G}$ also a chordless path on 7 vertices $P_7 = (3, 6, 1, 4, 7, 2, 5)$. Let us mark vertices of U by labels $1', 2', 3', 4', 5', 6', 7'$ along the path $P_7 = (3, 6, 1, 4, 7, 2, 5)$ starting from 3 to 5. Denote $S(T) = \{x \in VG - U : N_U(x) = T\}$ and $\tilde{S}(T) = \{x \in VG - U : \tilde{N}_U(x) = T\}$. In addition, we denote $S_W = S(\emptyset) \cap W$ and $S_B = S(\emptyset) \cap B$. Also, let us admit the following simplifications. We shall omit braces if they are inclosed by parentheses (for example, we shall write $S(2, 4)$ instead of $S(\{2, 4\})$), and we shall write $Star$ and Sun instead of $Star_{1,2,3}$ and Sun_4 , respectively.

In order to prove the theorem, we first deduce a number of claims. In these claims we use essentially the fact that graphs $Star$ and Sun are self-complementary (in the bipartite sense), and therefore $\tilde{G}$ is also $(Star, Sun)$ -free.

Claim 1. $S(4) = S(2, 6) = \emptyset$.

Proof. If $x \in S(4)$, then G contains induced $Star(1, 2, 3, 4, 5, 6, x)$. Hence $S(4) = \emptyset$. By analogy, $\tilde{S}(4') = \emptyset$ and consequently $S(2, 6) = \emptyset$ since $S(2, 6) = \tilde{S}(4')$.

Claim 2. $S(3) = S(5) = S(1, 5) = S(3, 7) = S(1, 3, 5) = S(3, 5, 7) = \emptyset$.

Proof. If a vertex $x \in VG - U$ is adjacent to vertices 1 and 5 and nonadjacent to vertex 7, then G contains induced $Star(2, 1, x, 5, 6, 7, 4)$. Hence $S(1, 5) = S(1, 3, 5) = \emptyset$. By symmetry, $S(3, 7) = S(3, 5, 7) = \emptyset$, and by analogy, $\tilde{S}(3', 5', 7') = \emptyset$. Since $S(3) = \tilde{S}(3', 5', 7')$, we have $S(3) = \emptyset$, and by symmetry, $S(5) = \emptyset$.

Claim 3. $S(3, 5) = \emptyset$.

Proof. Suppose $x \in S(3, 5)$. Then, since G is prime, there must be a vertex y with exactly one neighbour in set $\{4, x\}$. Without loss of generality, let y be adjacent to 4 but not x . Taking into account that $S(4) = \emptyset$, y must have a neighbour in set $\{2, 6\}$. With regard to symmetry, we may assume, without loss of generality, that y is adjacent to 2. But then G contains either induced $Star(1, 2, y, 6, 5, x, 7)$ (if y is adjacent to 6) or induced $Star(2, y, 4, 5, 6, 7, x)$ (if y is not adjacent to 6), a contradiction.

Claim 4. $S(1, 3) = S(5, 7) = \emptyset$.

Proof. Suppose $x \in S(1, 3)$. Since G is prime, we may assume without loss of generality that there exists a vertex y adjacent to 2 but not x . Then y is adjacent to 4, otherwise G contains either induced $Star(1, 2, y, 6, 5, 4, 7)$ (if y is adjacent to 6) or induced $Star(6, 5, 4, 3, 2, y, x)$ (if y is not adjacent to 6). But then G contains either induced $Star(x, 1, 2, y, 6, 7, 4)$ (if y is adjacent to 6) or induced $Star(1, x, 3, 4, 5, 6, y)$ (if y is not adjacent to 6), a contradiction. Hence $S(1, 3) = \emptyset$ and by symmetry $S(5, 7) = \emptyset$.

Claim 5. $S(2) = S(6) = S(2, 4) = S(4, 6) = \emptyset$.

Proof. Suppose $x \in S(2)$. Since G is prime, we may assume without loss of generality that there exists a vertex y adjacent to 1 but not x . Then y is not adjacent to 3, otherwise either Claim 2 or Claim 3 is not valid with respect to another $P_7 = (x, 2, 3, 4, 5, 6, 7)$. Next, y is adjacent to 5 else G contains induced $Star(5, 4, 3, 2, 1, y, x)$. But then either Claim 2 (if $(y, 7) \notin EG$) or Claim 4 (if $(y, 7) \in EG$) is not valid with respect to the $P_7 = (x, 2, 3, 4, 5, 6, 7)$, a contradiction. Hence $S(2) = \emptyset$. In addition, we have by symmetry $S(6) = \emptyset$, and by analogy $\tilde{S}(2') = \tilde{S}(6') = \emptyset$. Since $S(2, 4) = \tilde{S}(2')$ and $S(4, 6) = \tilde{S}(6')$, hence the claim.

Claim 6.

- (a) If $x \in S(2, 4, 6)$ and $y \in S(1) \cup S(7) \cup S(1, 7)$, then x is adjacent to y .
- (b) If $x \in S_W$ and $y \in S(1, 5, 7) \cup S(1, 3, 7) \cup S(1, 7)$, then x is not adjacent to y .

Proof. If a vertex $x \in S(2, 4, 6)$ is not adjacent to a vertex $y \in S(1) \cup S(1, 7)$, then G contains induced $Star(5, 6, x, 2, 1, y, 3)$. The case with $y \in S(7)$ can be proved by analogy. Part (b) of the claim is a consequence of part (a) and the following equalities: $S_W = \tilde{S}(2', 4', 6')$, $S(1, 5, 7) = \tilde{S}(1')$, $S(1, 3, 7) = \tilde{S}(7')$ and $S(1, 7) = \tilde{S}(1', 7')$.

Claim 7. Graphs $G[S(2, 4, 6) \cup S_B]$ and $G[S_W \cup S(1, 3, 5, 7)]$ are $2K_2$ -free.

Proof. Suppose vertices $a, b \in S(2, 4, 6)$ and $c, d \in S_B$ induce in G a $2K_2$ with edges (a, c) and (b, d) . Then G contains induced $Sun(a, 2, b, 4, c, 1, d, 5)$, a contradiction. The second part of the claim follows from equalities $S_W = \tilde{S}(2', 4', 6')$ and $S_B = \tilde{S}(1', 3', 5', 7')$ and from the fact that $\widetilde{2K_2}$ is isomorphic to $2K_2$.

In the next claim we use the following well known fact concerning $2K_2$ -free bipartite graphs: vertices of each part of a $2K_2$ -free bipartite graph can be linearly ordered under inclusion of their neighbourhoods.

Claim 8. Let a be a vertex in $S(2, 4, 6)$ such that $N_{S_B}(b) \subseteq N_{S_B}(a)$ for any $b \in S(2, 4, 6)$. Then vertex a is adjacent to all the vertices in S_B .

Proof. Assume, to the contrary, that S_B contains a vertex x nonadjacent to a . Without loss of generality we shall suppose that x is a nearest to set U vertex in S_B which is not adjacent to a . Denote a shortest path connecting x to a vertex $i \in U$ by $P_{xi} = (x = x_0, x_1, x_2, \dots, i)$. Due to the choice of a , x has no neighbours in $S(2, 4, 6)$, otherwise $N_{S_B}(b) \not\subseteq N_{S_B}(a)$ for any vertex $b \in S(2, 4, 6)$ adjacent to x . Taking into account Claims 1 and 5, we conclude that $x_1 \in S_W$. Due to Claim 6(b), $x_2 \notin S(1, 3, 7) \cup S(1, 5, 7) \cup S(1, 7)$.

Suppose that $x_2 \in S_B$. Then clearly x_2 is adjacent to a , otherwise we have a contradiction with the choice of x , since x_2 is situated nearer to U than x . But then G contains induced $Star(x_0, x_1, x_2, a, 2, 1, 4)$, a contradiction.

Suppose next $x_2 \in S(1)$, then, by Claim 6(a), G contains induced $Star(3, 4, a, x_2, x_1, x_0, 1)$, a contradiction. By analogy, $x_2 \notin S(7)$.

Taking into account Claims 2, 3, 4, we conclude now that $x_2 \in S(1, 3, 5, 7)$. Without loss of generality let us suppose that $N_{S_W}(z) \subseteq N_{S_W}(x_2)$ for any $z \in S(1, 3, 5, 7)$.

Since $\tilde{G}$ is connected, there must be a vertex y in W nonadjacent to x_2 . If $y \in S(2, 4, 6)$, then G contains induced $Star(y, 6, 7, x_2, x_1, x_0, 3)$. Hence $y \in S_W$. Assume without loss of generality that y is a nearest to set U vertex in S_W which is not adjacent to x_2 . Denote a shortest path connecting y to a vertex $j \in U$ by $P_{yj} = (y = y_0, y_1, y_2, \dots, j)$.

Due to the assumption concerning the choice of vertex x_2 , $y_1 \notin S(1, 3, 5, 7)$, otherwise $N_{S_W}(y_1) \not\subseteq N_{S_W}(x_2)$. Due to claim 6(b), $y_1 \notin S(1, 3, 7) \cup S(1, 5, 7) \cup S(1, 7)$. In addition, we can conclude that $y_1 \notin S(1) \cup S(7)$. Indeed, if $y_1 \in S(1)$, then G contains induced $Star(4, 5, x_2, 1, y_1, y_0, 2)$, and similarly if $y_1 \in S(7)$. Thus $y_1 \in S_B$.

First let us state that $y_1 \neq x_0$, otherwise G contains induced $Star(y_0, x_0, x_1, x_2, a, 2, 7)$. Next we conclude that y_1 is not adjacent to a , otherwise G contains either induced $Star(y_0, y_1, a, x_2, x_1, x_0, 7)$ (if y_1 is not adjacent to x_1) or induced $Sun(a, y_1, x_1, x_2, 2, y_0, x_0, 7)$ (if y_1 is adjacent to x_1). Consequently, due to the choice of a , y_1 has no neighbours in $S(2, 4, 6)$. Thus $y_2 \in S_W$. It follows from the assumption concerning vertex y_0 that y_2 is adjacent to x_2 , but then G contains induced $Star(y_0, y_1, y_2, x_2, 1, 2, 5)$. This contradiction completes the proof of the claim.

Claim 9. *Let x be a vertex in S_W such that $N_{S(1,3,5,7)}(x) \subseteq N_{S(1,3,5,7)}(y)$ for any $y \in S_W$. Then vertex x has no neighbours in set $S(1, 3, 5, 7)$.*

Proof. The statement follows directly from Claim 8 and equalities $S_W = \tilde{S}(2', 4', 6')$ and $S_B = \tilde{S}(1', 3', 5', 7')$.

Claim 10. *If $S(1, 3, 7) = \emptyset$ and $S(1, 5, 7) = \emptyset$, then $S(2, 4, 6) = \emptyset$.*

Proof. Suppose $S(2, 4, 6) \neq \emptyset$ and let a be a vertex in $S(2, 4, 6)$ such that $N_{S_B}(b) \subseteq N_{S_B}(a)$ for any vertex $b \in S(2, 4, 6)$. Consider a shortest path $P_{ai} = (a = a_0, a_1, a_2, \dots, i)$ connecting vertex a to a vertex $i \in U$ in graph $\tilde{G}$. Without loss of generality let us assume that for any vertex $b \in S(2, 4, 6)$ with

$N_{S_B}(b) = N_{S_B}(a)$, a shortest path connecting b to a vertex in U is not shorter than P_{ai} .

It follows from Claims 2, 3, 4, 6(a), 8 and the hypothesis of the claim that a_1 belongs to $S(1, 3, 5, 7)$. Suppose first that $a_2 \in S(2, 4, 6)$. Due to the choice of a_0 , we have $N_{S_B}(a_2) \neq N_{S_B}(a_0)$. Let b be a vertex in S_B adjacent to a_2 in $\tilde{G}$, then $\tilde{G}$ contains induced $Star(a_0, a_1, a_2, b, 3, 6, 7)$, a contradiction.

Suppose next that $a_2 \in S_W$. Without loss of generality, we may assume that for any vertex $b \in S_W$, $N_{S(1,3,5,7)}(b) \subseteq N_{S(1,3,5,7)}(a_2)$ in graph $\tilde{G}$. By Claim 9, a_2 has no neighbours in set $S(1, 3, 5, 7)$ in graph G . In addition, by Claim 6(a), a_2 has no neighbours in set $S(1, 3, 7) \cup S(1, 5, 7) \cup S(1, 7)$ in G . Finally, a_2 has no neighbours in set $S_B \cup S(1) \cup S(7)$ in G , otherwise G would contain induced $Star(a_1, 3, 4, a_0, b, a_2, 6)$ for any vertex $b \in S_B \cup S(1) \cup S(7)$ adjacent to a_2 . But then a_2 is isolated in G , a contradiction.

Claim 11. *If $S(1, 3, 7) = \emptyset$ and $S(1, 5, 7) = \emptyset$, then $S(1, 3, 5, 7) = \emptyset$.*

Proof. Suppose $S(1, 3, 5, 7) \neq \emptyset$. It follows from Claim 10 that $S_W \neq \emptyset$, otherwise any vertex in $S(1, 3, 5, 7)$ is isolated in graph $\tilde{G}$. Let a be a vertex in S_W such that $N_{S(1,3,5,7)}(a) \subseteq N_{S(1,3,5,7)}(b)$ for any vertex $b \in S_W$. Consider a shortest path $P_{ai} = (a = a_0, a_1, a_2, \dots, i)$ connecting vertex a to a vertex $i \in U$ in graph G . Without loss of generality let us assume that for any vertex $b \in S_W$ with $N_{S(1,3,5,7)}(b) = N_{S(1,3,5,7)}(a)$, a shortest path connecting b to a vertex in U is not shorter than P_{ai} .

By Claim 9, vertex $a = a_0$ has no neighbours in set $S(1, 3, 5, 7)$. Hence $a_1 \notin S(1, 3, 5, 7)$. In addition, $a_1 \notin S(1) \cup S(7)$. Indeed, if $a_1 \in S(1)$, then G contains induced $Star(4, 5, b, 1, a_1, a_0, 2)$ with $b \in S(1, 3, 5, 7)$, and similarly if $a_1 \in S(7)$. Thus, by Claims 2, 3, 4, 6(a), $a_1 \in S_B$ and consequently, by Claims 1, 5, 10, $a_2 \in S_W$. Due to the choice of a_0 , we must assume that a_2 has a neighbour b in $S(1, 3, 5, 7)$. But then G contains induced $Star(a_0, a_1, a_2, b, 1, 2, 5)$, a contradiction.

Claim 12. $|S(1)| \leq 1$, $|S(7)| \leq 1$, $|S(1, 7)| \leq 1$, $|S(1, 3, 7)| \leq 1$, $|S(1, 5, 7)| \leq 1$.

Proof. Suppose $x, y \in S(1)$ or $x, y \in S(1, 7)$. Then Claim 5 is violated with respect to $P_7 = (x, 1, 2, 3, 4, 5, 6)$. Hence $|S(1)| \leq 1$ and $|S(1, 7)| \leq 1$. By symmetry, $|S(7)| \leq 1$. Finally, since $S(1, 3, 7) = \tilde{S}(7')$ and $S(1, 5, 7) = \tilde{S}(1')$, we have $|S(1, 3, 7)| \leq 1$, $|S(1, 5, 7)| \leq 1$.

Now, to conclude the theorem, let us consider the following alternative cases that exhaust all possibilities for G .

Case 1: $S(1, 7) \neq \emptyset$. Let $S(1, 7) = \{x\}$. Then $S(1) = \emptyset$, otherwise G contains induced $Star(4, 3, 2, 1, x, 7, y)$ with $y \in S(1)$. By symmetry, $S(7) = \emptyset$. Since $S(1, 3, 7) = \tilde{S}(7')$ and $S(1, 5, 7) = \tilde{S}(1')$, we have $S(1, 3, 7) = \emptyset$, $S(1, 5, 7) = \emptyset$. Therefore, due to Claims 10 and 11, $S(2, 4, 6) = \emptyset$ and $S(1, 3, 5, 7) = \emptyset$. Taking into account Claims 1-5, 6(b), we conclude that the vertices in set $U \cup \{x\}$ have

no neighbours outside of $U \cup \{x\}$. This means by virtue of connectivity of G that $VG = U \cup \{x\}$ and hence $G = C_8$, i.e. G is $K_{1,3}$ -free.

Case 2: $S(1) \neq \emptyset$ or $S(7) \neq \emptyset$. Let $S(1) = \{x\}$. Then $S(1, 3, 7) = \emptyset$ (else G contains induced $Star(5, 6, 7, y, 1, x, 3)$ with $y \in S(1, 3, 7)$) and $S(1, 5, 7) = \emptyset$ (else G contains induced $Star(6, 7, y, 1, 2, 3, x)$ with $y \in S(1, 5, 7)$). Also, by symmetry, $S(7) \neq \emptyset$ implies $S(1, 3, 7) = \emptyset$ and $S(1, 5, 7) = \emptyset$. Consequently, due to Claims 10 and 11, $S(2, 4, 6) = \emptyset$ and $S(1, 3, 5, 7) = \emptyset$. Thus, all the vertices in set U have degree at most 2. Moreover, it is not difficult to see that path $P_7 = (x, 1, 2, 3, 4, 5, 6)$ also satisfies conditions of Case 2, and hence vertex x has degree at most 2 as well. Applying similar arguments by induction, we deduce that all the vertices of G are of degree at most 2. Hence G is $K_{1,3}$ -free.

Case 3: $S(1, 3, 7) \neq \emptyset$ or $S(1, 5, 7) \neq \emptyset$. In this case, the complement to G satisfies conditions of Case 2, since $S(1, 3, 7) = \tilde{S}(7')$ and $S(1, 5, 7) = \tilde{S}(1')$. Therefore, $\tilde{G}$ is $K_{1,3}$ -free.

Case 4: If G does not satisfy conditions of the previous cases, then obviously $VG = U$ and hence $G = P_7$, i.e. G is $K_{1,3}$ -free.

The theorem is proved.

3 Clique-width of $Star_{1,2,3}$ -Free and Sun_4 -Free Bipartite Graphs

In this section we use the obtained characterization to show that the clique-width of graphs in the class under consideration is at most five. This fact in combination with the results of [3] leads to polynomial time algorithms for a number of problems which are NP-complete for general bipartite graphs.

Graphs of clique-width at most k were introduced in [2] as graphs which can be defined by k -expressions based on graph operations which use k vertex labels. To introduce the operations, let us define a k -graph as a labeled graph with vertex labels in $\{1, 2, \dots, k\}$. For k -graphs G and H with $VG \cap VH = \emptyset$, we denote by $G \oplus H$ the disjoint union of G and H . For a k -graph G , we denote by $\eta_{i,j}(G)$, where $i \neq j$, the k -graph obtained by connecting all the vertices labeled i to all the vertices labeled j in G . For a k -graph G , we denote by $\rho_{i \rightarrow j}(G)$ the k -graph obtained by the renaming of i into j in G . For every vertex v of a graph G and $i \in \{1, \dots, k\}$, we denote by $i(v)$ the k -graph consisting of one vertex v labeled by i .

With every graph G one can associate an algebraic expression which defines G built using the 3 types of operations mentioned above. We call such an expression a k -expression defining G if all the labels in the expression are in $\{1, \dots, k\}$. For example, graph consisting of 2 isolated vertices x and y can be defined by 1-expression $1(x) \oplus 1(y)$, and graph consisting of two adjacent vertices x and y can be defined by 2-expression $\eta_{1,2}(1(x) \oplus 2(y))$.

The clique-width of a graph G , denoted $cwd(G)$, is defined by:

$$cwd(G) = \min\{k : G \text{ can be defined by a } k\text{-expression}\}.$$

For determination of the clique-width of a graph, the following simple lemmas are useful.

Lemma 1. *Let $G_1, \dots, G_k$ be connected components of a graph G , then*

$$cwd(G) = \max_{1 \leq i \leq k} \{cwd(G_i)\}.$$

Lemma 2. *Let H be a maximal prime induced subgraph of a graph G , then*

$$cwd(G) = cwd(H).$$

Proof. Lemma 1 is obvious. To prove Lemma 2, let us first note that a maximal prime induced subgraph of a graph is unique up to isomorphism and has exactly one vertex in each set of vertices with the same neighbourhood. In addition, each such a set induces in the graph an empty subgraph. Hence, we can derive a k -expression defining G from a k -expression T defining H as follows. Suppose a vertex x of H appears in the k -expression T with a label j , and let $x_1, x_2, \dots, x_l$ be the vertices of G having the same neighbourhood in G as x . Replace subexpression $j(x)$ of T by expression $j(x_1) \oplus j(x_2) \oplus \dots \oplus j(x_l)$. Performing the same with each vertex of H , we obtain a k -expression defining G . Hence, $cwd(G) \leq cwd(H)$. The converse inequality is obvious.

Due to Lemmas 1 and 2, all graphs considered in this section will be prime and connected. In addition to the above general concepts, let us define some specific notions for bipartite graphs. We shall call a k -expression defining a bipartite graph $G = (W, B, E)$ *proper* if for any $a \in W$ and $b \in B$, the label of a is not equal to the label of b .

The proper clique-width of a bipartite graph G , denoted $pcwd(G)$, is defined by:

$$pcwd(G) = \min\{k : G \text{ can be defined by a proper } k\text{-expression}\}.$$

Clearly $cwd(G) \leq pcwd(G)$. We shall prove that $pcwd(G) \leq 5$ for any $(Star, Sun)$ -free bipartite graph $G = (W, B, E)$. First, let us prove this for the case when the bipartite complement to G is connected. Then, by Theorem 1 and the above assumption, G or $\tilde{G}$ is $K_{1,3}$ -free. Clearly, any connected $K_{1,3}$ -free bipartite graph is either a chordless cycle or a chordless path. Without loss of generality we may suppose that such a cycle or a path has at least 7 vertices, otherwise G is a single-vertex graph (see the proof of Theorem 1).

Proper 5-Expression Procedure for an Even Cycle

Input: a cycle $G = (c_1, \dots, c_{2n})$ with $n > 3$.

Output: A 5-expression T defining G .

1. Set $T = \eta_{3,4}(4(c_4) \oplus \eta_{2,3}(3(c_3) \oplus \eta_{1,2}(1(c_1) \oplus 2(c_2))))$.
2. For $i = 3, \dots, n$ do
 set $T = \rho_{5 \rightarrow 3}(\eta_{4,5}(4(c_{2i}) \oplus (\rho_{4 \rightarrow 2}(\eta_{4,5}(5(c_{2i-1}) \oplus T))))$.
3. Set $T = \eta_{1,4}(T)$.

It is not hard to verify that steps 1 and 2 of the above procedure define a chordless path with even number of vertices. Moreover, replacing step 3 by

3. Set $T = \eta_{4,5}(5(c_{2n+1}) \oplus T)$,

we get a procedure defining a chordless path with odd number of vertices. Thus, we have proved

Lemma 3. *If G is a $K_{1,3}$ -free bipartite graph, then $pcwd(G) \leq 5$.*

Now let us describe a proper 5-expression procedure defining the bipartite complement to a connected $K_{1,3}$ -free bipartite graph.

Proper 5-Expression Procedure for the Bipartite Complement to an Even Cycle

Input: the bipartite complement to a cycle $G = (c_1, \dots, c_{2n})$ with $n > 3$.

Output: A 5-expression T defining $\tilde{G}$.

1. Set $T = 1(c_1) \oplus 2(c_2) \oplus 3(c_3) \oplus 4(c_4)$.
2. For $i = 3, \dots, n$ do
 set $T = \rho_{5 \rightarrow 3}(\eta_{4,3}(4(c_{2i}) \oplus (\rho_{4 \rightarrow 2}(\eta_{2,5}(5(c_{2i-1}) \oplus \eta_{1,4}(T))))))$.

In order to transform the above procedure into the one defining the bipartite complement to a chordless path P_k , it is enough to add to it either

3. Set $T = \eta_{1,4}(T)$

if $k = 2n$, or

3. Set $T = \eta_{2,5}(5(c_{2n+1}) \oplus \eta_{1,4}(T))$

if $k = 2n + 1$.

Hence we have proved

Lemma 4. *If G is a $K_{1,3}$ -free bipartite graph, then $pcwd(\tilde{G}) \leq 5$.*

Now, to attain the purpose of the section, it remains to consider the case when $\tilde{G}$ is disconnected.

Lemma 5. *Let G be a connected bipartite graph such that $\tilde{G}$ is disconnected, and let $G_1, \dots, G_p$ be induced subgraphs of G such that $\tilde{G}_1, \dots, \tilde{G}_p$ are the connected components of $\tilde{G}$. If for any $i = 1, \dots, p$, $pcwd(G_i) \leq k$ with $k \geq 4$, then $pcwd(G) \leq k$.*

Proof. To prove the lemma, let us consider a particular form of a proper k -expression defining a bipartite graph $G = (W, B, E)$ such that every vertex in W is labeled by 1 and every vertex in B is labeled by 2. We shall call such a form canonical. Obviously, any proper k -expression can be transformed into canonical form by the renaming operation $\rho_{i \rightarrow j}$. Thus, without loss of generality, we let $T_1, \dots, T_p$ be proper k -expressions defining $G_1, \dots, G_p$ in canonical form. A proper k -expression T defining G can be constructed by the following obvious procedure.

Proper k -Expression Procedure Defining G

Input: proper k -expressions ($k \geq 4$) $T_1, \dots, T_p$ defining $G_1, \dots, G_p$ in canonical form.

Output: A k -expression T defining G .

1. Set $T = \rho_{2 \rightarrow 4}(\rho_{1 \rightarrow 3}(T_1))$.
2. For $i = 2, \dots, p$ do
 set $T = \rho_{2 \rightarrow 4}(\rho_{1 \rightarrow 3}(\eta_{2,3}(\eta_{1,4}(T_i \oplus T))))$.

Hence $pcwd(G) \leq k$.

From Lemmas 1-5 and Theorem 1 we obtain by induction

Theorem 2. *The clique-width of $(Star_{1,2,3}, Sun_4)$ -free bipartite graphs is at most 5.*

For a class of graphs with clique-width at most k , Courcelle et al. present in [3] a number of optimization problems, which, given a graph G in the class and an $O(f(|VG|, |VE|))$ algorithm to construct a k -expression defining G , can be solved for G in time $O(f(|VG|, |VE|))$.

It is not hard to see that a 5-expression defining a $(Star_{1,2,3}, Sun_4)$ -free bipartite graph with n vertices can be obtained in $O(n^3)$ time. Moreover, with some care such an expression can be constructed in time $O(n^2 + nm)$, where m is the number of edges of the graph. In conjunction with the results of [3] this leads to polynomial algorithms for a number of problems which are NP-complete in general bipartite graphs (see [6] for a formal definition of these problems).

Corollary 1. *Given a $(Star_{1,2,3}, Sun_4)$ -free bipartite graph G with n vertices and m edges, one can solve the following problem for G in time $O(n^2 + nm)$: dominating set, induced path, unweighted steiner tree.*

References

1. D.G. Corneil, H. Lerchs and L.K. Stewart, Complement reducible graphs, Discrete Appl. Math. 3 (1981) 163-174.
2. B. Courcelle, J. Engelfriet and G. Rozenberg, Handle-rewriting hypergraphs grammars, J. Comput. System Sci. 46 (1993) 218-270.
3. B. Courcelle, J.A. Makowsky and U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, Theory Comput. Systems 33 (2000) 125-150.
4. J-L. Fouquet, V. Giakoumakis, H. Thuiller and F. Maire, On graphs without P_5 and $\overline{P_5}$, Discrete Math. 146 (1995) 33-44.
5. J-L. Fouquet and V. Giakoumakis, On semi- P_4 -sparse graphs, Discrete Math. 165/166 (1997) 277-230.
6. M.G. Garey and D.S. Johnson, Computers and Intractability: A guide to the theory of NP-completeness, Mathematical series, W.H. Freeman and Company, (1979).
7. V. Giakoumakis and J.-M. Vanherpe, Bi-complement reducible graphs, Advances in Appl. Math. 18 (1997) 389-402.

8. R. Jamison and S. Olariu, P_4 -reducible graphs – a class of uniquely representable graphs, *Studies in Appl. Math.* 81 (1989) 79-87.
9. R. Jamison and S. Olariu, A unique tree representation for P_4 -sparse graphs, *Discrete Appl. Math.* 35 (1992) 115-129.
10. G. Tinhofer, Strong tree-cographs are Birkhoff graphs, *Discrete Appl. Math.* 22 (1988/89) 275-288.

Automatic Graphs and Graph *D0L*-Systems

Olivier Ly

LaBRI, Université Bordeaux I
ly@labri.u-bordeaux.fr

Abstract. The concept of end is a classical mean of understanding the behavior of a graph at infinity. In this respect, we show that the problem of deciding whether an infinite automatic graph has more than one end is recursively undecidable. The proof is based on the analysis of some global topological properties of the configuration graph of a self-stabilizing Turing machine. Next, this result is applied to show the undecidability of connectivity of all the finite graphs produced by iterating a graph *D0L*-system. We also prove that the graph *D0L*-systems with which we deal can emulate hyperedge replacement systems for which the above connectivity problem is decidable.

Introduction

The concept of *automatic graph* [3,19,27,33,2] intends to define infinite graphs in some constructive way, i.e. in terms of finite state automata. It naturally arise when one looks at the definition of automatic groups (see [14]) from the point of view of graph theory. The property of being automatic for a finitely generated group can be directly expressed as a property of its Cayley graph. The concept of automatic graph consists in considering automatic structures as defining infinite graphs which are not necessarily Cayley graphs of some groups, dropping the symmetry properties implied by the group structure hypothesis.

Deterministic automatic graphs are definable up to isomorphism by weak monadic second-order logical formulae (cf. [33]). This allows us to understand how they fit in infinite graph theory. Indeed, this result together with [7, Thm 6.5], implies that deterministic automatic graphs of bounded tree-width are *equational* (see [30] for the definition of tree-width). Therefore, the deterministic automatic graphs of bounded tree-width and bounded degree are exactly the deterministic *context-free* graphs in the sense of [24]. Note that automatic graphs are not of bounded tree-width in general.

We deal with the notion of *end* of an infinite graph which is intended to capture the concept of “way to infinity” (see [1,12,17,23,24]). The main result of this paper concerns the effective computation of the number of ends:

Theorem 1. *The problem of deciding whether an automatic graph has more than one end is recursively undecidable.*

This result contributes in clearing up the decidability boundary of the problem of determining the number of ends of an infinite graph. In the case of the

context-free graphs which are Cayley graphs of groups, it was proved in [34] that the number of ends is computable. The group structure hypothesis is actually useless for this result: The space of ends of a context-free graph is homeomorphic to the boundary of a computable regular language; and hence, the number of ends of a context-free graph is effectively computable (cf. [22]). It turns out that the property of being of bounded tree-width draws for the moment a decidability boundary for the question considered here.

Theorem 1 applies to two domains: combinatorial group theory and graph transformation systems. On the first hand, the question of determining the number of ends of an automatic graph arises through the research of algorithms in combinatorial group theory. By Stallings's Theorem [35,12], the computation of the number of ends of a group, i.e. of its Cayley graph, is a step towards its decomposition as amalgamed product over finite subgroups. The study of effectiveness of Stallings Theorem led to a decomposition algorithm for context-free groups (cf. [34]). This algorithm can be extended to word hyperbolic groups in view of the recent result of [15] according to which the number of ends of such a group can be effectively computed. Note that context-free groups are word hyperbolic and that word hyperbolic groups are automatic [14,16]. A computer program was developed to compute the number of ends of an automatic group from an automatic structure ([4]). However, there is no known halting proof for this program; and identifying one-ended automatic groups in an effective way remains an open problem. Theorem 1 states that this task is not possible if one drops the group structure hypothesis.

On the other hand, Theorem 1 has an interpretation in graph transformation theory. Indeed, there is a connection between graph substitution systems, i.e. graph *DOL*-systems, and automatic graphs: The whole sequence of iterations of a graph *DOL*-system is described in terms of finite state automata by an automatic structure. More precisely, the sequence of finite graphs obtained in such a way actually is the sequence of the spheres of an automatic graph. The converse is true for a large class of automatic graphs to be defined in the text. Theorem 1 then turns out to be equivalent to the following result:

Theorem 2. *The problem of deciding whether all the finite graphs produced by iterating a graph *DOL*-system are connected, is undecidable.*

This contributes to the understanding of the decidability boundary of this problem: The graph *DOL*-systems to be considered here, which actually generalize *L*-systems [32], can emulate vertex replacement systems [10,13,8]; and for these last ones, the connectivity property described above was proved to be decidable.

Acknowledgements. The author is greatly indebted to D. B. A. Epstein for suggesting the problem and for many helpful discussions. The author also wants to thank his supervisor G. Sénizergues for pointing out the notion of self-stabilizing machine and also P. Narbel for improving Section 3.

1 Automatic Graphs

1.1 Definition

We deal with *labelled multi directed graphs* [9,6,31], i.e. tuple $(V, E, L, \text{vert}, \text{lab})$ where V is the set of vertices; E is the set of edges; $\text{vert} : E \rightarrow V \times V$ associates to each edge a directed pair of vertices. L is the set of labels which are defined by the map $\text{lab} : V \cup E \rightarrow L$. We consider graphs as metric spaces with the usual metric: edges are of length 1 and the distance between any two points is the length of a shortest path connecting them. For basics about formal languages and automata theory, the reader is referred to [18]. The concept, to be used here, of finite state automata recognizing directed pairs of words over an alphabet A is presented in [14, Chapter 1]. Such a device consists in a finite state automaton over $(A \cup \{\$ \}) \times (A \cup \{\$ \})$, where $\$$ is a *end-of-string* or *padding* symbol, whose language consists in some pairs of words (u, v) of $A^*.\$ \times A^*.\$$ of the same length with $u \in A^*$ or $v \in A^*$. By extension, we say that such an automaton recognizes a directed pair of words $(u, v) \in A^* \times A^*$ if it actually recognizes $(u.\$^{\max\{0, |v| - |u|\}}, v.\$^{\max\{0, |u| - |v|\}})$.

Definition 1 (Automatic Graphs).

An **automatic graph structure** is a tuple $(W, M_0, (M_\ell)_{\ell \in L})$ where

- W is a finite state automaton over a finite alphabet A with $\mathcal{L}(W) \subset A^*$ prefix-closed. A is parted in two subsets S and $\bar{S}$ which are in one-one correspondence: Each element s of S is associated to an element $\bar{s}$ of $\bar{S}$.
- M_0 is a finite state automaton recognizing pairs of words of A^* such that $\mathcal{L}(M_0)$ is a subset of $\mathcal{L}(W) \times \mathcal{L}(W)$ which is an equivalence relation.
- L is a finite set and for each $\ell \in L$, M_ℓ is a finite state automaton recognizing pairs of words of $\mathcal{L}(W)$.

An automatic graph structure defines a graph which is said to be **automatic** and whose edges are labelled on $S \cup L$:

- The set of vertices is defined to be $\mathcal{L}(W)/\mathcal{L}(M_0)$. As $\mathcal{L}(W)$ is prefix-closed, it contains the empty word; its class under $\mathcal{L}(M_0)$ is denoted by ε . Vertices are not labelled.
- The edges are some ordered pairs of words of $\mathcal{L}(W)$; they are of two kind: The first ones which are called **radial edges**, are the pairs of the form $(w, w.s)$ with $s \in S$ or $(w.\bar{s}, w)$ with $\bar{s} \in \bar{S}$; $(w, w.s)$ is labelled by s and connects the vertex represented by w to the vertex represented by $w.s$; $(w.\bar{s}, w)$ connects the vertex represented by $w.\bar{s}$ to the vertex represented by w , it is also labelled by s . Edges of the second kind which are called **transversal edges**, are the pairs of the form (w_1, w_2) which are recognized by some M_ℓ . Such an edge is labelled by ℓ and connects the vertex represented by w_1 to the vertex represented by w_2 .

Let us note that this definition of automatic graphs is more strict than the one of [19,2]; one can look at automatic graphs in the above sense as lying between Cayley graphs of automatic groups and automatic structures (cf. [3]).

Descriptions of graphs by automatic structures are *constructive* in the following sense: Let us consider an automatic graph structure for a graph. On the first hand, we can get a set of unique representatives of vertices as a regular language: Given a total ordering of A , let us consider the *shortlex* ordering of $\mathcal{L}(W)$ (see [18,14]). Each equivalence class of $\mathcal{L}(W)$ according to $\mathcal{L}(M_0)$ has a unique minimal word for this ordering which is called the shortlex minimal representative of the class in question. The language of shortlex minimal representatives is regular (see [14]). On the other hand, transversal edges can be recognized effectively. Note that this was not a priori clear in view of Definition 1. Each M_ℓ can indeed be transformed according to M_0 in such a way that the language it recognizes is closed under the equivalence relation defined by M_0 i.e. in such a way that it satisfy the following property: If $w_1, w_2, w'_1, w'_2 \in \mathcal{L}(W)$ are such that for $i = 1, 2$: w_i and w'_i are $\mathcal{L}(M_0)$ -equivalent i.e. $(w_i, w'_i) \in \mathcal{L}(M_0)$, then $(w_1, w'_1) \in \mathcal{L}(M_\ell)$ if and only if $(w_2, w'_2) \in \mathcal{L}(M_\ell)$ (see [14]).

Lots of examples of automatic graphs can be found in [14] as Cayley graphs of automatic groups which are automatic. For instance, free groups and free abelian groups give examples of trees and grids as automatic graphs as well as word hyperbolic groups. But the concept of automatic graph does not require such strong symmetry properties as the group framework does (see [3]). Indeed, automatic graphs are generally not homogeneous. For instance, regular trees (see [5] for a definition) and deterministic co-deterministic context-free graphs in the sense of [24] are automatic (see [33]).

1.2 Turing Machine Configuration Graph

One of the main tools involved in the proof of Theorem 1 is a slight variation of the concept of Turing machine configuration graph (see e.g. [26] for a definition of Turing machine configuration graph).

We deal with one tape deterministic Turing machines with *several* tape heads. Basics about Turing machines can be found e.g. in [18]. Let $T = (Q, \Gamma, \delta, q_0, F, \square)$ be a Turing machine with $n \geq 1$ tape heads; Q is the finite set of states, Γ is the tape alphabet, δ is the transition mapping, q_0 is the initial state, F is the set of final states and $\square$ is the blank symbol. Let $A = (Q \times \{1, \dots, n\}) \cup (\Gamma \setminus \square)$. Let $\mathcal{L} \subset A^*$ denote the language of instantaneous descriptions of T . Let us consider $\mathcal{L}.\square^*$ i.e. the language of words of the form $\text{ld}.\square\square\dots\square$ with $\text{ld} \in \mathcal{L}$. Note that a word of $\mathcal{L}.\square^*$ also encodes an instantaneous description of T in a non ambiguous way. Let $\overline{\mathcal{L}}.\square^*$ denote the prefix closure of $\mathcal{L}.\square^*$. We consider the graph G_T defined as follows: Vertices of G_T are the elements of $\overline{\mathcal{L}}.\square^*$. For all $w_1, w_2 \in \overline{\mathcal{L}}.\square^*$ and $a \in A$ such that $w_2 = w_1a$ there is an edge labelled by a connecting w_1 to w_2 . And for any two words $w_1, w_2 \in \overline{\mathcal{L}}.\square^*$ of the same length defining respectively two instantaneous descriptions ld_1 and ld_2 such that $\text{ld}_1 \xrightarrow{T} \text{ld}_2$, there is an edge connecting w_1 to w_2 . Note that as $\overline{\mathcal{L}}.\square^*$ is the prefix closure of $\mathcal{L}.\square^*$, some vertices of G_T do not define any instantaneous description.

The next lemma is deserved in [3] for the usual concept of Turing machine configuration graphs.

Lemma 1. G_T is automatic. Moreover, an automatic structure defining it can be effectively constructed from T .

Let v be a vertex of G_T . The distance between it and ε i.e. the minimal length of a non oriented path connecting them, is denoted by $|v|$. Note that it is simply the length of v as an element of $\mathcal{L}.\square^*$.

2 Undecidability of the Problem of Ends

Ends. The concept of *end* of an infinite graph can be understood as expressing the idea of “way to infinity” (see e.g. [1,12,17,23,24]). Let G be an infinite connected graph and let x_0 be a vertex of G . A non oriented infinite path $\rho = x_0x_1x_2\dots$ in G is called an *infinite branch* if the distance between x_0 and x_n is strictly increasing and thus tends to infinity. Two infinite branches are said to be equivalent if they are connected in the complementary of any ball centered at x_0 . An *end* of G is an equivalence class of infinite branch. In particular, G has only one end if and only if the complementary in G of any ball centered at x_0 is connected. Note that when we deal with the topology of graph, orientations of edges do not matter. So, the notion of connectivity means here that any two vertices are connected by a path of non oriented edges.

Self-Stabilizing Turing Machines. Let $T = (Q, \Gamma, \delta, q_{\text{init}}, \emptyset, \square)$ be a one tape head Turing machine without final state such that for all $q \in Q$, $a \in \Gamma$, $b \in \{0, 1\}$, $\delta(q, a, b)$ is defined. We describe in the following a Turing machine $\tilde{T}$ with two tape heads which simulates the computation of T ; $\tilde{T}$ will be called the *self-stabilizing* machine associated to T . One of the aims of the construction of $\tilde{T}$ is to be able to control its behavior on all its instantaneous descriptions and not only on those produced by a normal computation from the initial state. The simulation is done by computing step by step the successive instantaneous descriptions of T during its computation. They are stocked on the tape without repetition using a new symbol $\#$ to separate them. At each global loop, $\tilde{T}$ verifies that the list of instantaneous descriptions which is stocked on its tape indeed describes the computation of T ; if it does not, $\tilde{T}$ goes into a special state called the BUG state. When one deals with some Turing machine, one is usually not able to anticipate its behavior on any instantaneous description. Here we want the machine to drive itself aware that it is not in the right computation, i.e. the computation from the initial state. This is the role of the BUG state.

Because of the lack of space, we shall omit the formal construction of $\tilde{T}$; however, we state its main property: From any instantaneous description ld of $\tilde{T}$, either $\tilde{T}$ catches up with the normal behavior, those coming from its initial state, i.e. there exists an instantaneous description ld' of $\tilde{T}$ such that $\text{ld} \xrightarrow{*} \text{ld}'$ and $\tilde{q}_{\text{init}}^1 \tilde{q}_{\text{init}}^2 \xrightarrow{*} \text{ld}'$ where $\tilde{q}_{\text{init}}$ is the initial state of $\tilde{T}$, or ld drives $\tilde{T}$ into the BUG state after a finite computation.

Proof of Theorem 1 is as usual a reduction to the Turing machines halting problem. We start with a one head machine T which we slightly modify into a

machine still denoted by T such that this last one never stops and takes a bounded space on the tape if and only if the original machine stops after a finite computation. We then consider the self-stabilizing machine $\tilde{T}$ associated to it and finally the configuration graph $G_{\tilde{T}}$ of $\tilde{T}$ which we also slightly modify: We add to it an infinite branch to which every vertices representing instantaneous descriptions with BUG state are connected.

One shows that this graph has one and only one end if and only if the initial Turing machine does not stop, otherwise it has two ends. This implies Theorem 1.

3 Graph D0L-Systems and Automatic Graphs

3.1 Graph D0L-Systems

The graph transformations to be introduced here is a generalization to graphs of the concept of L -systems (cf. [32]); note that some efforts have been already made to extend them to n -dimensional structures, under the name of *map generating systems*, i.e. *M0L-systems* (cf. [21,11,20]). They was initially intended to simulate the process of growing generations of tissue layers in biology. The concept to be introduced here generalizes the concept of *ions* (cf. [28,29,25]). It consists in the simultaneous context-free replacement of all the vertices and all the edges.

Definition 2 (Graph D0L-Systems). *In this paper, a **graph D0L-system** δ is defined according to a 3-uple $(L, \ell_0, \Delta_V, \Delta_E)$ where:*

- $L = L_V \cup L_E$ is a finite set of label. L_V is the set of **vertex labels** and L_E is the set of **edge labels**; $\ell_0 \in L_V$ is the initial symbol.
- Δ_V is a mapping associating to each vertex label a finite graph whose vertices (respectively edges) are labeled over L_V (respectively L_E). These graphs are provided with an additional structure: A linear ordering of the vertex set.
- Δ_E is a mapping associating to each edge label a finite bipartite graph each vertex part of which is linearly ordered; the pair of vertex parts is directed itself. Edges are labeled on L_E and vertices are not labeled.

A graph is said to be a **δ -graph** if its vertices are labeled by elements of L_V and its edges by elements of L_E . Performing δ on a δ -graph $\Gamma = (V, E, L, \text{vert}, \text{lab})$ consists in doing the following parallel transformations on vertices and edges:

- Each vertex $v \in V$ is replaced by $\Delta_V(\text{lab}(v))$.
- Let $e \in E$ be an edge connecting a vertex v_1 to a vertex v_2 ; if the first vertex subset of $\Delta_E(\text{lab}(e))$ (the second one respectively) has as many vertices as $\Delta_V(\text{lab}(v_1))$ has ($\Delta_V(\text{lab}(v_2))$ respectively), then e is replaced by $\Delta_E(\text{lab}(e))$. The first vertex subset (the second one respectively) of this last one is glued to the vertices of the graph stemming from its origin v_1 (its target v_2 respectively) according to linear orderings. The edges which do not satisfy this condition are simply deleted.

Once simultaneous substitutions of all the vertices and all the edges are done, orderings become useless and are forgotten. A graph *D0L*-system defines a sequence of finite graphs $(\Gamma_n)_{n \geq 0}$ defined inductively by $\Gamma_0 = \Delta_V(\ell_0)$ and for all $n \geq 0$, $\Gamma_{n+1} = \delta(\Gamma_n)$; $(\Gamma_n)_{n \geq 0}$ is called the ***DOL***-sequence associated to δ .

Figure 1 describes an example of *D0L*-system which generates square grids.

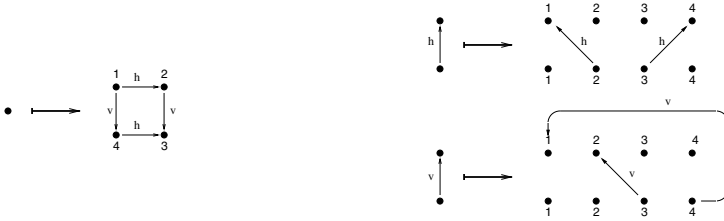


Fig. 1. A *D0L*-system which generates a sequence of grids. Note that vertex labels are not indicated as there is only one type of vertex

Graph *D0L*-systems as defined above generalize the classical *D0L*-systems over words; however one can wonder whether they can be defined in an algebraic way as word *D0L*-systems are. In this respect, let us note that graph *D0L*-systems are strictly more powerful than the *D0L*-systems which are associated to *HR* or *VR* graph grammars; the former ones indeed are able to generate grids of unbounded width while the last ones only generate graphs of bounded tree-width.

3.2 Connectivity Problem

We will see in this section how the notions of graph *D0L*-systems and of automatic graphs are connected which will lead us to show the equivalence between Theorem 1 and Theorem 2. We deal with a special class of automatic graphs satisfying the following conditions:

- The equivalence relation defining vertices is trivial, i.e. diagonal.
- Transversal edges only connect pairs of words of the same lengths.
- All the states of the transversal edge automata are final.

The sphere of radius n of an automatic graph satisfying these conditions is simply made of the vertices represented by words of length n and the transversal edges connecting them. The third condition assures such an edge leaves some traces, i.e. some edges which can be loops, in the spheres of lower radii.

The following result shows how automatic structures arise as a natural mean to describe *D0L*-sequences of finite graphs. This is a crucial point in making the connection between Theorem 1 and Theorem 2.

Lemma 2. *Up to a modification of the labels, the sequence of spheres of an automatic graph satisfying the above conditions is a D0L-sequence. Conversely, every D0L-sequence of finite graphs whose first one is made of only one vertex is the sequence of the spheres of an automatic graph.*

Let G be an automatic graph defined by an automatic structure $(W, M_0, \{M_\ell\}_\ell)$ satisfying the conditions of Lemma 2. One shows easily that if all the spheres of G are connected then G has only one end; in order to get the converse statement, i.e. Proposition 3 to be stated below, we need to cut the *finite branches* of G , i.e. the paths of radial edges in G which are not prefix of any infinite branch. Indeed, it may happen that some paths made of radial edges do not lead to any infinite branch. We can avoid this phenomenon by deleting all the states of W through which there is no infinite run; this can be done in an effective way by checking the loops in W and deleting all the states of W which do not lead to any loop. Let G' denotes the graph obtained after this transformation. It still is automatic; and now, we can state the following fact:

Lemma 3. *Keeping the above notations, G has one and only one end if and only if all the spheres of G' are connected.*

One shows that the automatic graph constructed from a Turing machine in the proof of Theorem 1 satisfies the two first conditions of Lemma 2. The third can be achieved by setting all the states of all the automata of the automatic structure to be final; one can verify that this does not modify the set of its ends. From this, we then construct the graph needed to apply Proposition 3; we finally get Theorem 2 which was announced in Introduction. Note that only the direct way of Proposition 3 is needed for that; nevertheless the full equivalence shows that Theorem 1 and Theorem 2 are in fact equivalent.

3.3 Emulation of VR Grammars by Graph D0L-Systems

Let $G = (L_V, L'_V, L_E, P, S)$ be a deterministic VR-grammar where L_V is the set of vertex labels, $L'_V \subset L_V$ is the set of *terminal* vertex labels, L_E is the set of edge label; P is a set of *production*, and $S \in L_V \setminus L'_V$ is the *initial non-terminal* (cf. [13, Chap. 1, Def. 1.3.5]). We consider the sequence $(\Gamma_n)_n$ of graphs defined by induction as following: Γ_0 is the graph of the production of G which is associated to S . For each $n \geq 0$, Γ_{n+1} is obtained from Γ_n by replacing each vertex labelled by a non-terminal for which a production is defined. As G is confluent, the ordering of the derivation steps in this transformation does not matter. A sequence of graphs obtained in such a way is called a *VR-sequence* of graphs.

Lemma 4. *Let G be a deterministic VR-grammar; let $(\Gamma_n)_n$ be the VR-sequence associated to it. Then all the graphs produced by G are connected if and only if for all $n \geq 0$, Γ_n is connected.*

The main decidable property of VR-languages of graphs is that their monadic second-order logic theories, with quantifications over vertex sets only, are

decidable (cf. [13, Chap. 1.4.2]). The connectivity property can be expressed by a monadic second-order logical formula whose quantifications only are over vertex sets; this implies in particular that there exists an algorithm to decide whether all the graphs of a VR-sequence are connected or not.

Proposition 1. *Up to a modification of the labeling, any VR-sequence of graph is a D0L-sequence.*

In view of Theorem 2 there is no hope of converse for this result; note that we could also see that seeing that graph *D0L*-systems can generate sequences of grids of unbounded width.

References

1. L. W. Ahlfors and L. Sario. *Riemann Surfaces*. Princeton University Press, 1960.
2. A. Blumensath and E. Grädel. Automatic structures. In *LICS'2000*, 2000.
3. A. Carbonne and S. Semmes. A graphic apology for symmetry and implicitness. Preprint.
4. A. Clow and S. Billington. Recognizing Two-Ended and Infinitely Ended Automatic Groups. Computer Program, Warwick University - UK, 1998.
5. B. Courcelle. Fundamental Properties of Infinite Trees. *Theoretical Computer Science*, 25:95–169, 1983.
6. B. Courcelle. Graph Rewriting: An algebraic and Logic Approach. In *Handbook of Theoretical Computer Science vol. B*, Van Leeuwen J. editor. Elsevier Science Publishers, 1990.
7. B. Courcelle. The Monadic Second-order Logic of Graphs IV : Definability Properties of Equational Graphs. *Annals of Pure and Applied Logic*, 49:193–255, 1990.
8. B. Courcelle. The Expression of Graph Properties and Graph Transformations in Monadic Second-Order Logic. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.
9. B. Courcelle and J. Engelfriet. A logical characterization of the sets of hypergraphs defined by hyperedge replacement grammars. *Math. Syst. Theory*, 28(6):515–552, 1995.
10. B. Courcelle, J. Engelfriet, and G. Rozenberg. Handle Rewriting Hypergraph Grammars. *JCSS*, 46:218–270, 1993.
11. M. De Does and A. Lindenmayer. Algorithms for the Generation and Drawing of Maps Representing Cell Clones. In *GGACS'82*, volume 153 of *Lect. Notes in Comp. Sci.*, pages 39–57. Springer-Verlag, 82.
12. W. Dicks and M.J. Dunwoody. *Groups Acting on Graphs*, volume 17 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1989.
13. J. Engelfriet and G. Rozenberg. Node Replacement Graph Grammars. In G. Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.
14. D.B.A. Epstein, J.W. Cannon, D.F. Holt, S.V.F. Levy, M.S. Paterson, and W.P. Thurston. *Word processing in groups*. Jones and Bartlett, 1992.
15. V. Gerasimov. Detecting Connectedness of the Boundary of a Hyperbolic Group. Preprint, 1999.
16. E. Ghys and P. de la Harpe. *Sur les groupes hyperboliques d'après Mikhael Gromov*, volume 83 of *Progress in Mathematics*. Birkhäuser, 1990.

17. R. Halin. Über Unendliche Wege in Graphen. *Math. Ann.*, 157:125–137, 1964.
18. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company, 1979.
19. B. Khoussainov and A. Nerode. Automatic Presentations of Structures. In *Logic and Computational Complexity (Indianapolis, IN, 1994)*, volume 960 of *Lecture Notes in Comput. Sci.*, pages 367–392. Springer, Berlin, 1995.
20. A. Lindenmayer. An Introduction to Parallel Map Generating Systems. In *GGACS'86*, volume 291 of *Lect. Notes in Comp. Sci.*, pages 27–40. Springer-Verlag, 1986.
21. A. Lindenmayer and G. Rozenberg. Parallel generation of maps: developmental systems systems for cell layers. In *GGACS'78*, volume 73 of *Lect. Notes in Comp. Sci.*, pages 301–316. Springer-Verlag, 1978.
22. O. Ly. On Effective Decidability of the Homeomorphism Problem for Non-Compact Surfaces. *Contemporary Mathematics - Amer. Math. Soc.*, 250:89–112, 1999.
23. W. S. Massey. *Algebraic Topology : An Introduction*, volume 56 of *Graduate Text in Mathematics*. Springer, 1967.
24. D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 37:51–75, 1985.
25. P. Narbel. *Limits and Boundaries of Word and Tiling Substitutions*. PhD thesis, Paris 7, 1993.
26. C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.
27. L. Pelecq. *Isomorphisme et automorphismes des graphes context-free, équationnels et automatiques*. PhD thesis, Bordeaux I, 1997.
28. J. Peyrière. Processus de naissance avec interaction des voisins, évolution de graphes. *Ann. Inst. Fourier, Grenoble*, 31(4):181–218, 1981.
29. J. Peyrière. Frequency of patterns in certain graphs and in Penrose tilings. *Journal de Physique*, 47:41–61, 1986.
30. N. Robertson and P. Seymour. Some New Results on the Well-Quasi Ordering of Graphs. *Annals of Discrete Math.*, 23:343–354, 1984.
31. G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1. World Scientific, 1997.
32. G. Rozenberg and A. Salomaa. *The Mathematical Theory of L-Systems*. Academic Press, 1980.
33. G. Sénizergues. Definability in weak monadic second-order logic of some infinite graphs. In *Dagstuhl seminar on Automata theory: Infinite computations. Wadern, Germany*, volume 28, pages 16–16, 1992.
34. G. Sénizergues. An effective version of Stallings' theorem in the case of context-free groups. In *ICALP'93*, pages 478–495. *Lect. Notes Comp. Sci.* 700, 1993.
35. J.R. Stallings. On torsion-free groups with infinitely many ends. *Ann. of Math.*, 88:312–334, 1968.

Bilinear Functions and Trees over the $(\max, +)$ Semiring

Sabrina Mantaci¹, Vincent D. Blondel², and Jean Mairesse¹

¹ LIAFA - Université Paris VII

Case 7014, 2 place Jussieu, 75251 Paris Cedex 05, France

`{mairesse, sabrina}@liafa.jussieu.fr`

² Department of Mathematical Engineering, CESAME - University of Louvain

Avenue Georges Lemaitre, 4, B-1348 Louvain-la-Neuve, Belgium

`blondel@inma.ucl.ac.be`

Abstract. We consider the iterates of bilinear functions over the semiring $(\max, +)$. Equivalently, our object of study can be viewed as recognizable tree series over the semiring $(\max, +)$. In this semiring, a fundamental result associates the asymptotic behaviour of the iterates of a linear function with the maximal average weight of the circuits in a graph naturally associated with the function. Here we provide an analog for the ‘iterates’ of bilinear functions. We also give a triple recognizing the formal power series of the worst case behaviour.

Remark. Due to space limitations, the proofs have been omitted. A full version can be obtained from the authors on request.

1 Introduction

Among all the complete binary trees having k internal nodes, what is the largest possible value of the difference between the number of internal nodes having even height and the number of leaves having odd height? As a byproduct of the tools developed in this paper, we will effectively solve this problem, see Example 5 below.

The $(\max, +)$ semiring has been studied in various contexts in the last forty years. It appears in Operations Research for optimization problems (see [9,14]); it is a useful tool to study some decision problems in formal language theory (see [16,17,21,22]); and it has an important role in the modelling and analysis of Discrete Event Systems (see [1,6,13]). In all of these applications, linear functions over the $(\max, +)$ semiring play a preeminent role. To give just one example, the dates of occurrence of events in a Timed Event Graph, a class of Discrete Event Systems, are given by the iterates of a linear function over the $(\max, +)$ semiring, see [1,6]. It is natural to study the direct generalization of linear functions: bilinear functions over the $(\max, +)$ semiring.

There is another possible way to introduce and motivate our study. Trees are one of the most important structure in computer science. They constitute a basic data structure; they are also the natural way to describe derivations of context

free grammars used by compilers. Formal power tree series with coefficients in a semiring were introduced by Berstel and Reutenauer in [2], and further studied for instance in [4,5]. In [2], the authors concentrate on recognizable series with coefficients in a field. They prove, among many other things, that the height of a tree defines a series which is not recognizable over a field (Example 9.2 in [2]). On the other hand, it is straightforward to show that this series is recognizable over the $(\max, +)$ semiring (Example 1 below). In this paper, we study the general class of recognizable tree series in one letter over the $(\max, +)$ semiring.

The $(\max, +)$ semiring $\mathbb{R}_{\max}$ is the set $\mathbb{R} \cup \{-\infty\}$, equipped with the $\max$ operation, written additively, i.e., $a \oplus b = \max\{a, b\}$, and the usual sum, written multiplicatively, i.e., $a \otimes b = a + b$. The neutral elements of the two operations are respectively $-\infty$ and 0. The $(\max, +)$ semiring is *idempotent*: $a \oplus a = a$, for all a . When there is no possible confusion, we simplify the notation by writing ab instead of $a \otimes b$. On the other hand, the operations denoted by $+$, $-$, $\times$ and $/$ always have to be interpreted in the conventional algebra.

We define accordingly the semimodule $\mathbb{R}_{\max}^n$ (for $u, v \in \mathbb{R}_{\max}^n$, $(u \oplus v)_i = u_i \oplus v_i$, and for $u \in \mathbb{R}_{\max}^n, \lambda \in \mathbb{R}_{\max}$, $(\lambda u)_i = \lambda u_i$). We denote by $(e_1, \dots, e_n)$ the canonical basis of $\mathbb{R}_{\max}^n$ (i.e. $(e_i)_i = 0$ and $(e_i)_j = -\infty$ for $j \neq i$).

A *linear form* of dimension n over $\mathbb{R}_{\max}$ is a function $f : \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}$ verifying $f(u \oplus v) = f(u) \oplus f(v)$, $\forall u, v \in \mathbb{R}_{\max}^n$, and $f(\lambda u) = \lambda f(u)$, $\forall u \in \mathbb{R}_{\max}^n, \lambda \in \mathbb{R}_{\max}$. A *linear function* of dimension n over $\mathbb{R}_{\max}$ is a function $f : \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}^n$ verifying the same two conditions. A *bilinear function* of dimension n over $\mathbb{R}_{\max}$ is a function $f : \mathbb{R}_{\max}^n \times \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}^n$, such that for all a in $\mathbb{R}_{\max}^n$ the functions $f_a, g_a : \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}^n$ defined by $f_a(x) = f(a, x)$, $g_a(x) = f(x, a)$ are linear.

As usual, we denote by f^k the k -th iterate of a function f . The asymptotic behavior of the iterates of a linear function is well understood. Indeed one of the most famous results in the $(\max, +)$ semiring (see for instance the textbooks [1, 19] and the references therein) is the following one. For all initial vector $\alpha \in \mathbb{R}_{\max}^n$ such that $\forall i, \alpha_i \neq -\infty$, and for all linear form $\beta : \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}$ such that $\forall i, \beta(e_i) \neq -\infty$, we have

$$\lim_k \frac{\beta[f^k(\alpha)]}{k} = \rho(f), \quad (1)$$

where $\rho(f)$ is the maximal average weight of the simple circuits in a graph canonically associated with f .

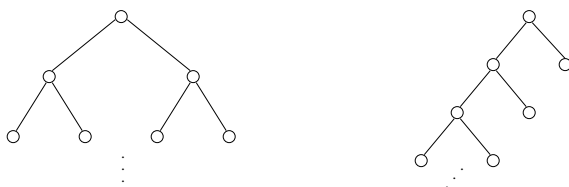
In this paper, we prove analog results for bilinear functions. Let f be a bilinear function of dimension n and let t be a (complete ordered binary) tree. We define recursively the functions $f^t : \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}^n$ by $f^t(u) = u$ when t is the root tree, and $f^t(u) = f(f^{t_1}(u), f^{t_2}(u))$ when t has t_1 and t_2 as left and right subtrees. Let the *size* $|t|$ of a tree t be the number of its internal nodes. Let us consider $\alpha \in \mathbb{R}_{\max}^n$ and a linear form $\beta : \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}$ such that $\forall i, \beta(e_i) \neq -\infty$. We prove the following result:

$$\limsup_k \frac{\max_{t, |t|=k} \beta[f^t(\alpha)]}{k} \quad (2)$$

does not depend on β and is equal to the maximal average weight of finitely many “simple” weighted trees. The quantity in (2) is called the *spectral radius* (of f at α). In comparison with the result for linear functions, note that the spectral radius depends on α . A tree attaining the maximum in $\max_{t, |t|=k} \beta[f^t(\alpha)]$ is called a *maximal tree* (of size k).

The computation of the spectral radius for bilinear functions gives rises to situations that are conceptually different from those for linear functions. In order to motivate the reader, and before unfolding our results, we illustrate some of these differences with two simple examples.

As a first example, consider the bilinear function f of dimension 2 defined by $f: (u_1, v_1)^T \times (u_2, v_2)^T \mapsto (u_2v_2 \oplus u_1v_2, Mu_1v_1)^T$, where $M \in \mathbb{R}$, and let $\alpha = (-1, 0)^T$ and β be such that $\beta(u) = u_1 \oplus u_2$. The spectral radius is $\max(0, M/3, 2M/3 - 1)$. Moreover, when $M < 0$, the branch trees (or ‘gourmand de la vigne’ according to [24], see the right of the figure) are the only maximal trees; when $M = 3$, all trees are maximal; when $M > 0, M \neq 3$, the maximal trees of size k are the ones with respectively no leaf, two leaves or one leaf of odd height if k equals 0,1 or 2 modulo 3 (left of the figure). Thus, although the dependence of the spectral radius on M is continuous, the trees that achieve maximality change drastically.



Now let $f(u, v) = (1u_2v_2, u_3v_3, u_4v_4, u_1v_1)^T$ be a bilinear function of dimension 4, and let $\alpha = (0, 0, 0, 0)^T$ and β be such that $\beta(e_i) \neq -\infty$ for all i . As will be easy to deduce from Proposition 1, the spectral radius is then equal to $8/15$, a value that was hard to guess from the coefficients of f and α . We finally illustrate the fact that the spectral radius generally depends on α . For this purpose, consider the same example as above but with $\alpha = (0, c, 0, 0)^T$ for $c \geq 0$. The spectral radius is then $8/15 + c$.

In the last part of the paper, we consider the formal power series over $\mathbb{R}_{\max}$:

$$S(\alpha, f, \beta) = \bigoplus_{k \in \mathbb{N}} \left(\bigoplus_{t, |t|=k} \beta[f^t(\alpha)] \right) x^k.$$

It follows from known results ([2,8,18,12]) that this series is recognizable. Here we provide an alternative proof of this result and we give an explicit construction of a triple recognizing $S(\alpha, f, \beta)$. This construction is a priori different from the known one.

All the results are presented for bilinear functions. This restriction is made only to simplify the presentation. The results are easy to adapt to the case of multilinear functions.

2 Trees

We consider complete ordered binary trees, that is trees where each node has no children or has a left and a right child. The formal definition of a binary tree that we consider is the one given in [20] or [23]. As usual we denote by A^* the free monoid over the set A .

Definition 1. An (unlabelled complete ordered binary) tree is a finite non-empty prefix-closed subset t of $\{0, 1\}^*$, such that if $v \in t$, then either both $v \cdot 0 \in t, v \cdot 1 \in t$, or both $v \cdot 0 \notin t, v \cdot 1 \notin t$. Let A be a finite alphabet. A (complete ordered binary) labelled tree over A is a partial mapping $\tau: \{0, 1\}^* \rightarrow A$ whose domain $\text{dom}(\tau)$ is a tree.

The definitions below are given for trees. They are easy to extend to labelled trees. Since a tree is a non-empty prefix-closed subset of $\{0, 1\}^*$, it always contains the empty word ε that is called the *root* of the tree. The tree $t = \{\varepsilon\}$ is called the *root tree*. The *frontier* of a tree t is the set $\text{fr}(t) = \{v \in t \mid v \cdot 0, v \cdot 1 \notin t\}$. The elements in t , $\text{fr}(t)$, and $t \setminus \text{fr}(t)$ are called respectively *nodes*, *leaves*, and *internal nodes*. The *size* of a tree t , denoted by $|t|$, is the number of its internal nodes. We denote by $\mathcal{T}$ the set of trees, and by $\mathcal{T}^n$ the set of trees of size n .

Given a tree t different from the root tree, we define its *left subtree* $t_1 = \{w \in \{0, 1\}^* \mid 0 \cdot w \in t\}$ and its *right subtree* $t_2 = \{w \in \{0, 1\}^* \mid 1 \cdot w \in t\}$, and we write $t = \varepsilon(t_1, t_2)$. In the case of a labelled tree τ with left and right subtrees τ_1 and τ_2 and with $\tau(\varepsilon) = a$, we write $\tau = a(\tau_1, \tau_2)$.

3 Linear Functions over the $(\max, +)$ Semiring

The $(\max, +)$ semiring has been defined in the introduction. We use the matrix and vector operations induced by the semiring structure: if A and B are two matrices of appropriate sizes with coefficients in the semiring $\mathbb{R}_{\max}$, we define $(A \oplus B)_{ij} = A_{ij} \oplus B_{ij} = \max(A_{ij}, B_{ij})$ and $(A \otimes B)_{ij} = \bigoplus_k A_{ik} \otimes B_{kj} = \max_k (A_{ik} + B_{kj})$. We still use the simplified notation AB for $A \otimes B$.

Let f be a linear function of dimension n over the $(\max, +)$ semiring, see §1. We associate canonically a matrix A to f by setting $A_{ij} = f(e_i)_j$. Then A^k is the matrix associated with f^k . Below, we have chosen to state the results in terms of powers of matrices instead of iterates of linear functions. All the results are classical; for details see the textbooks [1, 19, 15] and the references therein.

We associate with a square matrix A of dimension n , the valued directed graph $\mathcal{G}(A)$ with nodes $\{1, \dots, n\}$ and with an arc (i, j) if $A_{ij} \neq -\infty$, this arc being valued by A_{ij} . We use the classical terminology of graph theory. In particular, we use the notation $i \rightarrow j$ to denote the existence of a path from node i to node j in the graph.

Let us consider a triple (α, A, β) where $A \in \mathbb{R}_{\max}^{n \times n}$, $\alpha \in \mathbb{R}_{\max}^{1 \times n}$ and $\beta \in \mathbb{R}_{\max}^{n \times 1}$. We say that (α, A, β) is *trim* if for all k , there exist i, j , such that $\alpha_i \neq -\infty$, $\beta_j \neq -\infty$, $i \rightarrow k$ and $k \rightarrow j$. Given a matrix $A \in \mathbb{R}_{\max}^{n \times n}$, we define

$$\rho(A) = \bigoplus_{l=1, \dots, n} \bigoplus_{i_1, \dots, i_l} (A_{i_1 i_2} \otimes A_{i_2 i_3} \otimes \dots \otimes A_{i_l i_1})^{1/l}.$$

In words, $\rho(A)$ is equal to the maximal average weight of the circuits of $\mathcal{G}(A)$.

Theorem 1. *Let (α, A, β) be a trim triple. We have $\limsup_k (\alpha A^k \beta) / k = \rho(A)$.*

In particular, when A is irreducible, we have $\limsup_k A_{ij}^k / k = \rho(A)$ for all i, j . It is also well known that $\rho(A)$ is equal to the maximal eigenvalue of A (we say that $\lambda \in \mathbb{R}_{\max} \setminus \{-\infty\}$ is an *eigenvalue* of A if there exists $u \in \mathbb{R}_{\max}^n$ such that $A \otimes u = \lambda \otimes u$).

The result in Theorem 1 can be easily extended to a non-trim triple (α, A, β) . Indeed, if $(\tilde{\alpha}, \tilde{A}, \tilde{\beta})$ is the trim part of (α, A, β) , then we have $\forall k, \alpha A^k \beta = \tilde{\alpha} \tilde{A}^k \tilde{\beta}$.

4 Bilinear Functions over the $(\max, +)$ Semiring

A bilinear function of dimension n has the following structure:

$$B: \mathbb{R}_{\max}^n \times \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}^n, \quad (u, v) \mapsto \begin{pmatrix} \bigoplus_{i,j} B_{1ij} \otimes u_i \otimes v_j \\ \dots \\ \bigoplus_{i,j} B_{nij} \otimes u_i \otimes v_j \end{pmatrix},$$

where $B_{kij} \in \mathbb{R}_{\max}$ for all i, j, k .

We recall the recursive definition of $B^t: \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}^n$, for $t \in \mathcal{T}$.

- If $t = \{\varepsilon\}$ then $\forall u, B^t(u) = u$;
- if $t = \varepsilon(t_1, t_2)$ then $\forall u, B^t(u) = B(B^{t_1}(u), B^{t_2}(u))$.

Let us consider the triple (α, B, β) where $\alpha \in \mathbb{R}_{\max}^n$, $B: \mathbb{R}_{\max}^n \times \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}^n$ is a bilinear function, and $\beta: \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}$ is a linear form (see §1). The function *recognized* by the triple is the function $\mu: \mathcal{T} \rightarrow \mathbb{R}_{\max}$ defined by $\mu(t) = \beta[B^t(\alpha)]$.

Example 1. The height function $h: \mathcal{T} \rightarrow \mathbb{N}$ is defined recursively as follows: $h(\{\varepsilon\}) = 0$ and if $t = \varepsilon(t_1, t_2)$, then $h(t) = 1 + \max(h(t_1), h(t_2))$.

Consider the triple (α, B, β) defined as follows:

$$\alpha = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \quad B(u, v) = \begin{pmatrix} u_1 v_1 \\ 1u_1 v_2 \oplus 1u_2 v_1 \end{pmatrix}, \quad \beta(u) = u_2.$$

This triple recognizes the height function. Indeed, consider $t = \varepsilon(t_1, t_2)$ and assume that we have $B^{t_1}(\alpha) = (0, h(t_1))$ and $B^{t_2}(\alpha) = (0, h(t_2))$. Then it follows that $B^t(\alpha) = B((0, h(t_1)), (0, h(t_2))) = (0, 1h(t_1) \oplus 1h(t_2)) = (0, h(t))$.

Definition 2. *Given a triple (α, B, β) , we define its spectral radius as the quantity:*

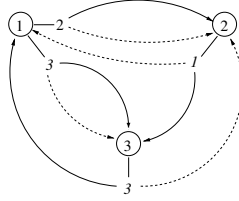
$$\limsup_k \max_{t \in \mathcal{T}^k} \frac{\beta[B^t(\alpha)]}{k}$$

Our goal is to study the spectral radius. To do this, we would like to associate with a bilinear function B a sort of graph describing it (mimicking the situation for linear functions).

5 Tree-Graphs, Tree-Paths and Tree-Circuits

We define a particular type of directed graph, denoted by $\mathcal{G}(B)$, as follows. The set of nodes of $\mathcal{G}(B)$ is $\{1, 2, \dots, n\}$ (where n is the dimension of B) and for each $B_{kij} \neq -\infty$ there exists in $\mathcal{G}(B)$ a pair of arcs $((k, i), (k, j))$ where (k, i) is the *left arc* and (k, j) is the *right arc*. The pair $((k, i), (k, j))$ is valued by B_{kij} . We say that $\mathcal{G}(B)$ is the *tree-graph associated with B* .

Example 2. Consider the bilinear function $B: \mathbb{R}_{\max}^3 \times \mathbb{R}_{\max}^3 \rightarrow \mathbb{R}_{\max}^3$ defined by: $B(u, v) = (2u_2v_2 \oplus 3u_3v_3, 1u_3v_1, 3u_1v_2)^T$. The associated graph has three nodes $\{1, 2, 3\}$. Moreover $B_{122} = 2$, $B_{133} = 3$, $B_{231} = 1$ and $B_{312} = 3$. The corresponding pairs of arcs are shown in the following figure: we draw a continuous line to denote the left arc and a dashed line to denote the right arc.



Remark 1. A triple (α, B, β) can be considered as a bottom-up tree automaton in one letter over $\mathbb{R}_{\max}$. The tree-graph $\mathcal{G}(B)$ can be considered as a visualization of the corresponding top-down tree automaton. An extensive account on tree automata can be found in [7].

We now define the notions of tree-path and tree-circuit in a tree-graph, generalizing the classical notions of path and circuit in a graph.

Definition 3. Let B be a bilinear function of dimension n . A tree-path over $\mathcal{G}(B)$ is a tree τ over the alphabet $\{1, 2, \dots, n\}$, different from the root tree and such that if $v \in \text{dom}(\tau) \setminus \text{fr}(\tau)$ and $\tau(v) = k$, $\tau(v \cdot 0) = i$ and $\tau(v \cdot 1) = j$, then $B_{kij} \neq -\infty$. A tree-circuit over $\mathcal{G}(B)$ is a tree-path where the root and at least one of the leaves have the same label.

We will denote by $\text{path}(B)$ the set of all tree-paths in $\mathcal{G}(B)$, and by $\text{circ}(B)$ the set of all tree-circuits in $\mathcal{G}(B)$.

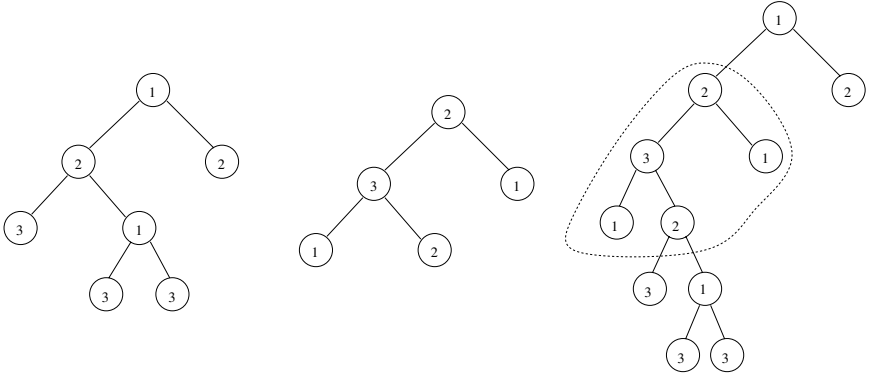
On the free monoid $\{0, 1\}^*$, we use the notation $\leq$ for the prefix order (with $0 \leq 1$). Let τ be a tree-path over $\mathcal{G}(B)$, and let $\tau(v) = i$ for some $v \in \text{dom}(\tau)$. Let c be a tree-circuit over $\mathcal{G}(B)$ whose root is labelled by i , and let y be a leaf of c labelled by i . The *composition of the tree-path τ with the tree-circuit c at the nodes v and y* is the tree-path $\tau[v, y]c$ defined as follows:

$$\text{dom}(\tau[v, y]c) = \{u \in \text{dom}(\tau) \mid v \not\leq u\} \cup \{vz \mid z \in \text{dom}(c)\} \cup \{vyw \mid vw \in \text{dom}(\tau)\}$$

$$(\tau[v, y]c)(u) = \begin{cases} \tau(u) & \text{if } v \not\leq u \\ c(w) & \text{if } u = vz \text{ and } z \in \text{dom}(c) \\ \tau(vw) & \text{if } u = vyw, vw \in \text{dom}(\tau) \end{cases}$$

To avoid ambiguities, it is necessary to use parentheses for iterated compositions. We omit parentheses when the composition is performed from left to right, for instance: $\tau[v, y]c[v', y']c' = (\tau[v, y]c)[v', y']c'$. When we do not need to specify about the exact positions where the composition is performed, we simplify the notation by writing $\tau[\cdot]c$ instead of $\tau[v, y]c$.

Example 3. Below we have represented a tree-path τ over the tree-graph of Example 2 (on the left of the figure) and a tree-circuit c over the same tree-graph (on the middle of the figure). The composition $\tau[0, 01]c$ produces the tree on the right of the figure.



Definition 4. A tree-path over $\mathcal{G}(B)$ is a simple tree-path if it can not be obtained by composition of a tree-path with a tree-circuit. A simple tree-circuit is a tree-circuit that can not be obtained by composition of two tree-circuits.

It follows from this definition that every tree-circuit can be written as an iterated composition of simple tree-circuits. Moreover every tree-path can be written as an iterated composition of a simple tree-path with a sequence of simple tree-circuits. We remark that tree-circuits and tree-paths can have several different decompositions.

Lemma 1. Let τ be a simple tree-path over $\mathcal{G}(B)$. Then we have $h(\tau) \leq n$ and $|\tau| \leq 2^n - 1$ (where h is the height function defined in Example 1). Let c be a simple tree-circuit, then we have $h(c) \leq 2n - 1$ and $|c| \leq n + n2^n$.

6 Asymptotic Behavior

Let us fix a vector α in $\mathbb{R}_{\max}^n$. We define a function $p_\alpha(\cdot)$, depending on B and α , on the set of trees labelled by $\{1, \dots, n\}$ as follows:

- if $\text{dom}(\tau) = \{\varepsilon\}$, and $\tau(\varepsilon) = k$ then $p_\alpha(\tau) = \alpha_k$;
- if $\tau = k(\tau_1, \tau_2)$, and $\tau_1(\varepsilon) = i$, $\tau_2(\varepsilon) = j$, then $p_\alpha(\tau) = B_{kij} + p_\alpha(\tau_1) + p_\alpha(\tau_2)$.

The easy but useful relation below provides an interpretation of $B^t(\alpha)$ in terms of maximally weighted tree-paths in $\mathcal{G}(B)$.

$$B^t(\alpha)_i = \max_{\tau \in \text{path}(B), \text{dom}(\tau)=t, \tau(\varepsilon)=i} p_\alpha(\tau). \quad (3)$$

Let c be a tree-circuit and let l be a leaf of c such that $c(l) = c(\varepsilon)$. The *average weight* of c is defined as

$$w_\alpha(c) = \frac{1}{|c|} \left(\sum_{u \in \text{dom}(c) \setminus \text{fr}(c)} B_{c(u)c(u,0)c(u,1)} + \sum_{u \in \text{fr}(c) \setminus \{l\}} \alpha_{c(u)} \right).$$

When $\alpha_{c(\varepsilon)} \neq -\infty$, then we have $w_\alpha(c) = (p_\alpha(c) - \alpha_{c(\varepsilon)})/|c|$.

A tree-circuit over $\mathcal{G}(B)$ is *maximal* if its average weight is greater than or equal to the average weight of any other tree-circuit over $\mathcal{G}(B)$. Since there is an infinite number of tree-circuits, the existence of maximal tree-circuits is not a priori guaranteed. We recall that $\text{circ}(B)$ is the set of the tree-circuits of $\mathcal{G}(B)$, and we denote by $\text{simp}(B)$ the set of the simple tree-circuits.

Lemma 2. *There exists a simple tree-circuit with maximal average weight, i.e.:*

$$\sup_{c \in \text{circ}(B)} w_\alpha(c) = \max_{c \in \text{simp}(B)} w_\alpha(c).$$

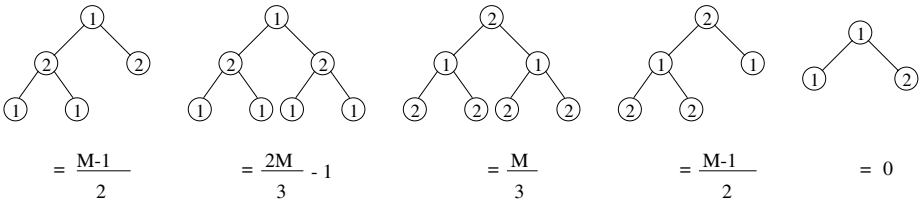
It follows from Lemma 1 and Lemma 2 that there always exist a maximal tree-circuit of height at most $2n - 1$.

The result below is to be compared with Theorem 1. We recall that given a triple (α, B, β) , we have defined its spectral radius in Definition 2.

Proposition 1. *Let us consider a triple (α, B, β) where $\alpha \in \mathbb{R}_{\max}^n$, $B: \mathbb{R}_{\max}^n \times \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}^n$ is a bilinear function, and $\beta: \mathbb{R}_{\max}^n \rightarrow \mathbb{R}_{\max}$ is a linear form such that $\alpha_i \neq -\infty$ and $\beta(e_i) \neq -\infty$ for all i . The spectral radius depends only on α and B , is denoted $\rho(\alpha, B)$, and is given by $\rho(\alpha, B) = \max_{c \in \text{simp}(B)} w_\alpha(c)$.*

The triple (α, B, β) is said to be *trim* if for all k in $\{1, \dots, n\}$, there exists a tree-path τ over $\mathcal{G}(B)$ such that $k \in \text{dom}(\tau)$, $\beta(e_{\tau(\varepsilon)}) \neq -\infty$, $p_\alpha(\tau) \neq -\infty$. Proposition 1 holds under the more general assumption that the triple is trim.

Example 4. Consider the bilinear function $f(u, v) = (u_2 v_2 \oplus u_1 v_2, M u_1 v_1)^T$, with $M \in \mathbb{R}$, and let $\alpha = (-1, 0)^T$, and β be such that $\beta(u) = u_1 \oplus u_2$ (same example as in the introduction). There are 9 simple tree-circuits. Only 5 among the 9 can have a maximal weight for some values of M . We have represented below these 5 simple tree-circuits, together with their corresponding average weight w_α .



We deduce the formula for the spectral radius given in the introduction: $\rho(\alpha, f) = \max(0, M/3, 2M/3 - 1)$.

7 Formal Power Series

Let us consider a triple (α, B, β) where $\alpha \in \mathbb{R}_{\max}^n$, B is a bilinear function of dimension n , and β is a linear form of dimension n . We consider the following formal power series in one indeterminate over the $(\max, +)$ semiring:

$$S(\alpha, B, \beta) = \bigoplus_{k \in \mathbb{N}} \left(\bigoplus_{t, |t|=k} \beta[B^t(\alpha)] \right) x^k. \quad (4)$$

For details concerning formal power series over a semiring, see [3,18]. A series S in one indeterminate over $\mathbb{R}_{\max}$ is *recognizable* if there exists an integer N and a triple (a, A, b) with $a \in \mathbb{R}_{\max}^{1 \times N}$, $A \in \mathbb{R}_{\max}^{N \times N}$, $b \in \mathbb{R}_{\max}^{N \times 1}$ and such that

$$S = \bigoplus_{k \in \mathbb{N}} (a \otimes A^k \otimes b) x^k \quad (5)$$

We also say that the triple (a, A, b) *recognizes* S . Using classical results, we obtain that the series $S(\alpha, B, \beta)$ is recognizable. Indeed, according to Theorem 7.1 in [2], the series $S(\alpha, B, \beta)$ is algebraic. Using an adaptation of an original argument by Parikh, see [8,18], an algebraic series in one indeterminate over a commutative and idempotent semiring is recognizable (the use of Parikh result in the context of $(\max, +)$ algebraic series appears in [12]).

Using the notions of simple tree-path and simple tree-circuit defined above, we obtain an alternative proof of this result. We get an explicit construction of a triple having the required property.

Proposition 2. *There exists a triple (a, A, b) of dimension $O(n2^{2n})$ which recognizes $S(\alpha, B, \beta)$.*

Example 5. In order to illustrate the expressiveness of these series, consider the following simple example. We define $f(u, v) = (Ku_2v_2, u_1v_1)^T$, and $\alpha = (0, M)^T$. Then it is easily seen that $f^t(\alpha)$ is equal to $(h$ being the height function):

$$\begin{pmatrix} K \times \#\{u \in t \setminus \text{fr}(t) \mid h(u) \text{ even}\} + M \times \#\{u \in \text{fr}(t) \mid h(u) \text{ odd}\} \\ K \times \#\{u \in t \setminus \text{fr}(t) \mid h(u) \text{ odd}\} + M \times \#\{u \in \text{fr}(t) \mid h(u) \text{ even}\} \end{pmatrix}.$$

For instance, if we choose $K = 1, M = -1$ and $\beta(u) = u_1$, the k -th coefficient in the resulting series $S(\alpha, f, \beta)$ is equal to the largest possible difference between the number of internal nodes of even height and leaves of odd height in trees of size k . The complete version of Proposition 2 (i.e. with proof) gives an explicit construction for computing these quantities for all k .

Acknowledgement. During the preparation of this article, the authors realized that Proposition 1 can be viewed as a particular case of a similar result proved by Stéphane Gaubert [11] on the growth of coefficients in an algebraic series over the $(\max, +)$ semiring. The authors would like to thank S. Gaubert and the four anonymous referees for corrections and suggestions which have greatly improved the paper.

References

1. F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity*. John Wiley & Sons, New York, 1992.
2. J. Berstel and C. Reutenauer. Recognizable formal power series on trees. *Theoretical Computer Science*, 18:115–148, 1982.
3. J. Berstel and C. Reutenauer. *Rational Series and their Languages*. Springer Verlag, 1988.
4. S. Bozapalidis. Constructions effectives sur les séries formelles d'arbres. *Theoretical Computer Science*, 77(3):237–247, 1990.
5. S. Bozapalidis. Convex algebras, convex modules and formal power series on trees. *Autom. Lang. Comb.*, 1(3):165–180, 1996.
6. G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot. A linear system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing. *IEEE Trans. Automatic Control*, 30:210–220, 1985.
7. H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on: <http://www.grappa.univ-lille3.fr/tata>, 1997.
8. J.H. Conway. *Regular algebra and finite machines*. Chapman and Hall, 1971.
9. R. Cuninghame-Green. *Minimax Algebra*, volume 166 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, 1979.
10. S. Gaubert. On rational series in one variable over certain dioids. Technical Report 2162, INRIA, 1994.
11. S. Gaubert. Personal communication, 1998.
12. S. Gaubert and J. Mairesse. Task resource models and $(\max, +)$ automata. In J. Gunawardena, editor, *Idempotency*, volume 11, pages 133–144. Cambridge University Press, 1998.
13. S. Gaubert and J. Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE Trans. Aut. Cont.*, 44(4):683–698, 1999.
14. M. Gondran and M. Minoux. *Graphs and Algorithms*. John Wiley & Sons, 1986.
15. J. Gunawardena, editor. *Idempotency*. Publications of the Newton Institute. Cambridge University Press, 1998.
16. K. Hashigushi. Limitedness theorem on finite automata with distance functions. *J. Computer System Sci.*, 24:233–244, 1982.
17. D. Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *Int. J. of Algebra and Computation*, 4(3):405–425, 1994.
18. W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Springer, 1986.
19. V. Maslov and S. Samborskii, editors. *Idempotent Analysis*, volume 13 of *Adv. in Sov. Math.* AMS, 1992.
20. M. Nivat. Binary tree codes. In *Tree automata and languages*, pages 1–19. Elsevier, 1992.
21. J.E. Pin. Tropical semirings. In J. Gunawardena, editor, *Idempotency*, pages 50–69. Cambridge University Press, 1998.
22. I. Simon. The nondeterministic complexity of a finite automaton. In M. Lothaire, editor, *Mots, Mélanges offerts à M.P. Schützenberger*, pages 384–400. Hermes, Paris, 1990.
23. W. Thomas. Automata on infinite objects. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, pages 133–192. Elsevier and MIT Press, 1990.
24. G.X. Viennot. Trees. In M. Lothaire, editor, *Mots*, pages 265–297, Paris, 1990. Hermès.

Derivability in Locally Quantified Modal Logics via Translation in Set Theory^{*}

Angelo Montanari¹, Alberto Policriti¹, and Matteo Slanina²

¹ Dipartimento di Matematica e Informatica, Università di Udine
Via delle Scienze 206, 33100 Udine, Italy

² Department of Computer Science, Stanford University
Stanford, CA 94305-9045, USA

Abstract. Two of the most active research areas in automated deduction in modal logic are the use of translation methods to reduce its derivability problem to that of classical logic and the extension of existing automated reasoning techniques, developed initially for the propositional case, to first-order modal logics. This paper addresses both issues by extending the translation method for propositional modal logics known as $\Box$ -as-Pow (read “box-as-powerset”) to a widely used class of first-order modal logics, namely, the class of *locally quantified* modal logics. To do this, we prove a more general result that allows us to separate (classical) first-order from modal (propositional) reasoning. Our translation can be seen as an example application of this result, in both definition and proof of adequateness.

1 Introduction

The existence of a large variety of different modal logics makes the traditional approach to theorem proving, based on the design of an efficient prover for every singular logic, infeasible apart for the few most widely used logics. This fact and the availability of a large set of deduction tools for classical first-order logic have produced a deep interest in methods to translate one logic into another (preserving validity). Many different translation methods for propositional modal logics (*relational*, *functional*, *semi-functional* translations, see [11, 12]) have been proved competitive with native methods explicitly tailored for particular modal logics. Their main weakness is the lack of complete generality, as they can be directly applied only to frame-complete modal logics whose characteristic class of frames is expressible in first-order logic. Even when a logic has the above characteristics, the problem of finding the corresponding class of frames, given an axiomatic description of the modal logic in a Hilbert style, is not trivial [6].

^{*} The first author was partially supported by the MURST project *Software Architectures and Languages to Coordinate Distributed Mobile Components*. The second author was partially supported by the CNR project *log(S.E.T.A.): Specifiche Esequibili e Teorie degli Aggregati. Metodologie e Piattaforme*.

The $\Box$ -as-*Pow* translation, proposed in [4], addressed these problems by translating modal formulas into set-theoretic terms. Such a translation is adequate for the class of all frame complete modal logics and the input to the translation consists of Hilbert style axioms, which solves the above correspondence problem. The $\Box$ -as-*Pow* translation has been extended to polymodal, temporal, and other extended modal logics [3, 4, 9] and generalized to non frame complete logics, thus achieving full generality for the class of finitely axiomatizable normal modal logics [2]. Recently, using similar techniques, Ohlbach found a way to incorporate into his translations axioms and rules of Hilbert calculi, thus creating a hybrid system able to deal with the axioms that cannot be directly captured by the translation [13].

“Modal logic” is usually considered a synonym for *propositional* modal logic. The introduction of first-order modal logics, strongly encouraged by the computer science community as a convenient tool for representating and reasoning about dynamic processes (e. g. computations), gives rise to a large number of variations, only a few of which are interesting for practical purposes. A broad survey is given in [7]. Here we choose one possible extension, by far the most used: a system with rigid designators and fixed domains. Following Garson’s classification we call these **Q1** type logics.

Even these logics, though, are usually too complex for automated deduction tools to deal with, so that when it comes to implementation some sort of reduction to the propositional case is usually employed. For example, most of the work on concurrent program verification and model checking is based on quantifier-free temporal logics, therefore only notationally different from propositional ones. However, a few exceptions do exist. Among the most significant ones there are the so-called *locally quantified* modal logics allowing one to use explicit quantification, but prevent modal operators from occurring in the scope of quantifiers. This is the case, for instance, of Manna and Pnueli’s temporal framework [8], where only such a restricted form of explicit quantification is allowed.

We first prove a general result showing that validity for a locally quantified formula can be *separated* into a classical (*first-order*) part and a modal (*propositional*) part. Next, we use this result to generalize the $\Box$ -as-*Pow* translation to locally quantified formulas, where the underlying frame structure is, as in the propositional case, any frame complete modal logic.

2 The $\Box$ -as-*Pow* Translation

Let us begin by quickly reviewing the propositional $\Box$ -as-*Pow* translation, giving its definition, a short discussion of the basic inspiring ideas, and the statement of the soundness and completeness theorem for frame-complete logics. For the proofs see [4].

We will use a fairly standard syntax for propositional modal logic, consisting of propositional variables (or symbols) $Q_1, Q_2, \dots$, logical connectives $\vee, \neg$, and

the unary modal operator $\Box$. Derived symbols, to be used as abbreviations, are $\Diamond$ (defined as $\neg\Box\neg$), $\wedge$, $\rightarrow$, and $\leftrightarrow$. Well-formed formulas are defined as usual.

The basic idea of the set-theoretic translation is simply to replace the accessibility relation R of the Kripke semantics with the membership relation $\in$ (in fact, $\in^{-1}$). A world v accessible from w becomes an *element* of w and a further step from v , using the accessibility relation R , will amount to *looking into* v in order to reach for one of its elements. As interesting consequences we have that worlds, frames, and valuations of propositional variables all become simply *sets* (of worlds), and that a frame $\mathcal{F}$ can be identified with its support $\mathcal{W}$, being the accessibility relation implicitly defined as the membership relation on $\mathcal{W}$. Moreover, since we clearly want all worlds v accessible from a given world w in a frame $\mathcal{W}$ to turn out themselves to be elements of $\mathcal{W}$, it is natural to require that all frames are *transitive* sets.

Since a valuation for a propositional variable is nothing but a set of worlds, the standard definition of $\models$ will allow us to associate a set of worlds to each propositional formula. This set, inductively defined on the structural complexity of the formula, will be the collection of those worlds in the frame in which the formula holds. Let φ be a propositional modal formula built on the propositional variables $Q_1, \dots, Q_n$, and let x be a variable used to denote the set $\mathcal{W}$ of all possible worlds, $y_1, \dots, y_n$ distinct variables (denoting the sets of worlds in which each Q_i is true). The $\Box$ -as-*Pow* or *set-theoretic* translation of φ is a term $\text{ST}(\varphi)$, inductively defined as follows:

$$\begin{aligned} \text{ST}(Q_i) &= y_i ; \text{ST}(\varphi \vee \psi) = \text{ST}(\varphi) \cup \text{ST}(\psi) ; \text{ST}(\neg\varphi) = x \setminus \text{ST}(\varphi) ; \\ \text{ST}(\Box\varphi) &= \text{Pow}(\text{ST}(\varphi)). \end{aligned}$$

By applying the $\Box$ -as-*Pow* translation, the fact that a formula φ is satisfied in the frame $\mathcal{W}$, with respect to a given valuation for $Q_1, \dots, Q_n$, amounts to saying that the substitution of $\mathcal{W}$ for x and of Q_i for y_i satisfies $x \subseteq \text{ST}(\varphi)$ (remember that $y_1, \dots, y_n$ are free in $\text{ST}(\varphi)$). To say that a formula φ is satisfied in $\mathcal{W}$ corresponds to saying that $\forall \vec{y}(x \subseteq \text{ST}(\varphi))$, where $\vec{y}$ stands for $y_1, \dots, y_n$. Finally, the fact that φ is valid is stated as $\forall x, \vec{y}(x \subseteq \text{ST}(\varphi))$. We shall later use a variation of this translation, that we denote by ST_C (the C is for “constant”), that uses constant symbols $c_1, \dots, c_n$ instead of the variables $y_1, \dots, y_n$. It is only used to simplify notation, being nothing but a skolemization of the original version.

The next step is to provide a set theory which allows one to use the $\Box$ -as-*Pow* translation to prove modal theorems. The first fact that must be taken into account is that $\in$ cannot have properties which can’t be guaranteed also for a generic accessibility relation R , hence it can be neither acyclic nor extensional, and a “minimalist” approach to axiomatic set theory becomes a *necessity*. The axiomatic set theory that has been introduced for this purpose (to be called Ω) is extremely simple. Its axioms are, essentially, the definitions of the set-theoretic operators employed in the $\Box$ -as-*Pow* translation:

$$\begin{aligned} x \in y \cup z &\leftrightarrow x \in y \vee x \in z ; & x \subseteq y &\leftrightarrow \forall z(z \in x \rightarrow z \in y) ; \\ x \in y \setminus z &\leftrightarrow x \in y \wedge x \notin z ; & x \in \text{Pow}(y) &\leftrightarrow x \subseteq y. \end{aligned}$$

When a modal logic is given by a finite set of Hilbert axioms, whose conjunction we denote by γ , and the logic is frame-complete, we define $\text{Axiom}(\gamma)$ as $\forall z(x \subseteq \text{ST}(\gamma))$, and the following result (soundness and completeness of the translation) holds ($\text{Tr}(x)$ stands for $\forall y(y \in x \rightarrow y \subseteq x)$, transitivity of x):

Theorem 1 $\vdash_{\gamma} \varphi \Leftrightarrow \Omega \vdash \forall x (\text{Tr}(x) \wedge \text{Axiom}(\gamma) \rightarrow (x \subseteq \text{ST}_C(\varphi)))$.

3 First-Order Modal Logics

We now review the definition of **Q1** quantified modal logics. See [5, 7] for details. We define the *language* of a first-order modal logic as given by: an infinite set of *variables*; a set of *constant symbols*; a set of *function symbols*; a set of *relation symbols*; the *boolean connectives* $\vee$ and $\neg$; the *modal operator* $\Box$; the *universal quantifier* $\forall$; the *auxiliary symbols* (and). Derived connectives and well-formed formulas are defined in the standard way.

Definition 2 Let **L** be a propositional modal logic (we view a logic as a set of formulas — its theorems). The *semantics of the Q1-L system* of quantified modal logic with rigid designators and fixed domain based on **L** is defined as follows.

A (**Q1**) *frame* is a triple $\mathfrak{F} = (\mathcal{W}, R, \mathcal{D})$, where $\mathcal{W}$ is a non empty set of *worlds*, $R \subseteq \mathcal{W} \times \mathcal{W}$ is the *accessibility (reachability) relation*, and $\mathcal{D} \neq \emptyset$ is the *domain of interpretation*. A (*rigid*) *interpretation* on $\mathfrak{F}$ is a triple $(\mathcal{F}_C, \mathcal{F}_F, \mathcal{F}_R)$, where:

- $\mathcal{F}_C$ assigns an element $\mathcal{F}_C(c)$ of $\mathcal{D}$ to each constant symbol c ;
- $\mathcal{F}_F$ assigns a function $\mathcal{F}_F(f) : \mathcal{D}^k \rightarrow \mathcal{D}$ to each k -ary function symbol f ;
- $\mathcal{F}_R$ assigns a subset $\mathcal{F}_R(p, a)$ of $\mathcal{D}^k$ to each k -ary relation symbol p and world $a \in \mathcal{W}$.

A (**Q1**) *model* is a 6-tuple $\mathcal{M} = (\mathcal{W}, R, \mathcal{D}, \mathcal{F}_C, \mathcal{F}_F, \mathcal{F}_R)$ such that $\mathfrak{F} = (\mathcal{W}, R, \mathcal{D})$ is a (**Q1**) frame and $(\mathcal{F}_C, \mathcal{F}_F, \mathcal{F}_R)$ is an interpretation on $\mathfrak{F}$. We say that $\mathcal{M}$ is *based on* $\mathfrak{F}$.

An *assignment* s is a function mapping each variable x_i to an element $s(x_i)$ of $\mathcal{D}$. We define $s(t)$ for a term t and truth/validity of a formula in the obvious ways. Note that $s(t)$ does not depend on a world — this is the meaning of the expression *rigid interpretation*. $\mathfrak{F}$ is a **Q1-L** *frame* if it validates all the formulas of **L**; $\mathcal{M}$ is a **Q1-L** *model* if it is based on a **Q1-L** frame. We use the symbols $a, s \models_{\mathbf{Q1-L}} \varphi$ to denote the truth of a formula φ in a world a under interpretation s , $\mathcal{M} \models_{\mathbf{Q1-L}} \varphi$ to denote its truth in a model $\mathcal{M}$, and $\models_{\mathbf{Q1-L}} \varphi$ to denote its validity in all the **Q1-L** frames, and so on. When no confusion can arise, we write **L** instead of **Q1-L**.

Let **L** be a propositional modal logic characterized by a Hilbert system. A Hilbert calculus for **Q1-L**, based on **L**, can be defined in such a way that the following soundness and completeness theorem holds.

Theorem 3 *If $\mathbf{L}$ is a frame-complete propositional modal logic and φ is a first-order modal formula, then $\vdash_{\mathbf{Q1-L}} \varphi \iff \models_{\mathbf{Q1-L}} \varphi$.*

As our subsequent proofs are entirely model-theoretical, we omit the details of the calculus. They can be found, with different flavors, in [5, 7].

4 Locally Quantified Formulas

In this section, we formally define locally quantified formulas and then prove some basic results that will play a major role in the generalization of the $\Box$ -as-*Pow* translation to them.

Definition 4 A first-order modal formula F is *locally quantified* if it is of the form $F = \psi\{Q_1/\varphi_1, \dots, Q_n/\varphi_n\}$, where ψ is a propositional modal formula over the variables $Q_1, \dots, Q_n$ and $\varphi_1, \dots, \varphi_n$ are closed classical formulas, that is, a locally quantified formula can be obtained from a propositional one by substituting all the propositional variables with closed classical formulas.

Let us prove now our basic separation result: to prove F , it is sufficient to take the φ_i 's, compute all their boolean combinations, find the valid ones, and use propositional forms of these, discarding the φ_i 's, as assumptions in a proof of the modal part ψ of F .

Let L be the first-order modal language of F . We denote by $\tilde{L}$ the language obtained from L by the addition of n propositional variables (0-ary relation symbols) distinct from those of L . In what follows, $\{\vec{Q} \leftrightarrow \vec{\varphi}\}$ will always be an abbreviation for $\{Q_1 \leftrightarrow \varphi_1, \dots, Q_n \leftrightarrow \varphi_n\}$, $\{\vec{Q}/\vec{\varphi}\}$ for the substitution $\{Q_1/\varphi_1, \dots, Q_n/\varphi_n\}$, $\mathbf{L}$ a frame-complete propositional modal logic.

Lemma 5 $\models_{\mathbf{Q1-L}} F \iff \{\vec{Q} \leftrightarrow \vec{\varphi}\} \models_{\mathbf{Q1-L}} \psi$.

Proof. Immediate once we observe that $Q_1, \dots, Q_n$ are not in the language of F . ■

For any given sequence of closed classical formulas $\varphi_1, \dots, \varphi_n$ ($\vec{\varphi}$ for short), let Σ be the set of all the classical propositional formulas σ over variables $Q_1, \dots, Q_n$ such that $\models \sigma\{\vec{Q}/\vec{\varphi}\}$ ($\sigma\{\vec{Q}/\vec{\varphi}\}$ is valid).

Lemma 6 *Let V be a truth-value assignment to $Q_1, \dots, Q_n$ such that, for each $\sigma \in \Sigma$, $V(\sigma) = \text{true}$. Then there exists a model $\mathfrak{A}$, based on the Herbrand preinterpretation of (the classical component of) $\tilde{L} \cup \{c_\alpha \mid \alpha < \omega\}$ (where the c_α are infinitely many new constant symbols), such that $\mathfrak{A} \models \{\vec{Q} \leftrightarrow \vec{\varphi}\}$ and, for each $i \in \{1, \dots, n\}$, $\mathfrak{A} \models Q_i$ if and only if $V(Q_i) = \text{true}$.*

Proof. Let

$$\begin{aligned} \theta_i &= \begin{cases} Q_i & \text{if } V(Q_i) = \text{true}, \\ \neg Q_i & \text{if } V(Q_i) = \text{false}, \end{cases} \\ \theta &= \theta_1 \wedge \dots \wedge \theta_n. \end{aligned}$$

It is easy to see that the request $\mathfrak{A} \models Q_i \iff V(Q_i) = \text{true}$, for $i \in \{1, \dots, n\}$, is equivalent to $\mathfrak{A} \models \theta$. Furthermore, even if we restrict to the given preinterpretation, there exists a model $\mathfrak{A} \models \{\vec{Q} \leftrightarrow \vec{\varphi}\}$ such that $\mathfrak{A} \models \theta$ if and only if there is a model $\mathfrak{A}$ of classical first-order logic such that $\mathfrak{A} \models \theta\{\vec{Q}/\vec{\varphi}\}$.

Let us assume now, for the sake of contradiction, that $\theta\{\vec{Q}/\vec{\varphi}\}$ does not have models based on the given Herbrand preinterpretation. It follows that $\theta\{\vec{Q}/\vec{\varphi}\}$ has no model (it is unsatisfiable), hence $\neg\theta\{\vec{Q}/\vec{\varphi}\}$ is valid, and so $\neg\theta \in \Sigma$. But then, by hypothesis, $V(\neg\theta) = \text{true}$, thus $V(\theta) = \text{false}$, contradicting the very definition of θ . ■

Lemma 7 $\models_{\mathbf{Q1-L}} F \iff \Sigma \models_{\mathbf{L}} \psi$.

Proof. By Lemma 5, $\models_{\mathbf{Q1-L}} F$ can always be rewritten as $\{\vec{Q} \leftrightarrow \vec{\varphi}\} \models_{\mathbf{Q1-L}} \psi$.

For the “if” part, assume $\Sigma \models_{\mathbf{L}} \psi$ and let $\mathcal{A}$ be an $\mathbf{L}$ -model of $\{\vec{Q} \leftrightarrow \vec{\varphi}\}$. Since $\{\vec{Q} \leftrightarrow \vec{\varphi}\} \models_{\mathbf{Q1-L}} \Sigma$, we have that $\mathcal{A}$ is an $\mathbf{L}$ -model of ψ .

Vice versa, let M be a propositional $\mathbf{L}$ -model such that $M \models \Sigma$ and $M \not\models \psi$. We build a first-order $\mathbf{L}$ -model $\mathcal{A} = (\mathcal{W}, R, \mathcal{D}, \mathcal{F}_C, \mathcal{F}_F, \mathcal{F}_R)$, with $\mathcal{W}$, R , and the interpretation of the Q_i ’s being the same as in M , such that $\mathcal{A} \models \{\vec{Q} \leftrightarrow \vec{\varphi}\}$.

For each $w \in \mathcal{W}$, Lemma 6 guarantees the existence of a model $\mathfrak{B}_w$ such that, for each $i \in \{1, \dots, n\}$, $w \models Q_i \iff \mathfrak{B}_w \models \varphi_i$.

Furthermore, all the $\mathfrak{B}_w$ ’s are based on the same preinterpretation. Then, letting (in $\mathcal{A}$) $\mathcal{W}$, R and the interpretation of the Q_i ’s be as in M , the domain and the interpretation of the constant and function symbols of L as in the Herbrand preinterpretation of $\tilde{L} \cup \{c_\alpha \mid \alpha < \omega\}$, and, for each world w , the interpretation of the relation symbols as in $\mathfrak{B}_w$, we obtain that $\mathcal{A}$ is an $\mathbf{L}$ -model such that $\mathcal{A} \models \{\vec{Q} \leftrightarrow \vec{\varphi}\}$ and $\mathcal{A} \not\models \psi$. ■

Corollary 8 *There is a finite set Σ of classical propositional formulas over $Q_1, \dots, Q_n$ such that, for all $\sigma \in \Sigma$, $\models \sigma\{\vec{Q}/\vec{\varphi}\}$, and $\models_{\mathbf{Q1-L}} F \iff \Sigma \models_{\mathbf{L}} \psi$.*

Proof. We can take one representative out of each equivalence class of the Σ given by Lemma 7 modulo logical equivalence. The cardinality of the set of these equivalence classes can be no more than 2^{2^n} . ■

An interesting problem to look at is whether a stricter bound on the cardinality of a Σ sufficient to prove F can be found.

5 Two-Sorted Translation of Locally Quantified Formulas

We now describe and prove the faithfulness of our translation in a two-sorted language. Let $F = \psi\{\vec{Q}/\vec{\varphi}\}$ be a locally quantified L -formula and $\mathbf{L}$ be a frame-complete modal logic axiomatized by the propositional modal formula γ .

We define L' as the two sorted language with sorts **term** and **set** and the following symbols:

- constant and function symbols of L are introduced in L' with sort **term**;
- for each n -ary relation symbol p of L we introduce in L' an $n+1$ -ary relation symbol of sort $\mathbf{term}^n \times \mathbf{set}$, denoted by the same symbol p ;
- the symbols $\in, \subseteq, \setminus, \cup$, and Pow of sort **set**;
- the constant symbols $c_1, c_2, \dots$, distinct from those of L , of sort **set**.

The theory Ω_2 , over the language L' , is axiomatized by the universal closure of the following formulas, where all the free variables are of sort **set**:

$$\begin{aligned} x \in y \cup z &\leftrightarrow x \in y \vee x \in z \\ x \in y \setminus z &\leftrightarrow x \in y \wedge \neg x \in z \\ x \subseteq y &\leftrightarrow \forall z : \mathbf{set} (z \in x \rightarrow z \in y) \\ x \in Pow(y) &\leftrightarrow x \subseteq y \end{aligned}$$

Definition 9 Fix a variable x of sort **set**. The translation of a (locally quantified) formula $F = \psi\{Q_1/\varphi_1, \dots, Q_n/\varphi_n\}$, where $t_1, \dots, t_n$ are terms of L and y is a variable of sort **set**, is expressed as follows:

$$\begin{aligned} \overline{(p(t_1, \dots, t_n))}(y) &= p(t_1, \dots, t_n, y), \quad \overline{(\theta \vee \eta)}(y) = \bar{\theta}(y) \vee \bar{\eta}(y), \quad \overline{(\neg \theta)}(y) = \neg \bar{\theta}(y), \\ \overline{(\forall z \theta)}(y) &= \forall z : \mathbf{term} \bar{\theta}(y), \\ \text{Axiom}^2(\gamma) &= \forall x_1, \dots, x_m : \mathbf{set} (x \subseteq \text{ST}(\gamma)), \\ \text{ST}^2(\varphi_i) &= \forall y : \mathbf{set} (y \in x \rightarrow (y \in c_i \leftrightarrow \bar{\varphi}_i(y))), \\ (\gamma, F)^{*2} &= \forall x : \mathbf{set} (\text{Trans}(x) \wedge \text{Axiom}^2(\gamma) \wedge \bigwedge_{i=1}^n \text{ST}^2(\varphi_i) \rightarrow x \subseteq \text{ST}_C(\psi)). \end{aligned}$$

Theorem 10 [Completeness of the translation] $\vdash_\gamma F \implies \Omega_2 \vdash (\gamma, F)^{*2}$.

Proof. If $\vdash_\gamma F$, the completeness of the **Q1-L** calculus implies that $\models_{\mathbf{Q1-L}} F$, Corollary 8 that $\Sigma \models_{\mathbf{L}} \psi$, where $\sigma \in \Sigma \implies \models \sigma\{\vec{Q}/\vec{\varphi}\}$, and hence, from the completeness of the propositional version of **L**, $\Sigma \vdash_\gamma \psi$. By the Global Deduction Theorem for modal logic, now, there is $k \in \mathbb{N}$ such that $\vdash_\gamma \bigwedge_{i=0}^k \Box^i \Sigma \rightarrow \psi$, where $\Box^i \Sigma = \{\Box^i \sigma \mid \sigma \in \Sigma\}$. Therefore, by the Completeness Theorem for the propositional ST_C translation,

$$\Omega \vdash \forall x \left(\text{Tr}(x) \wedge \text{Axiom}(\gamma) \rightarrow x \subseteq \text{ST}_C \left(\bigwedge_{i=0}^k \Box^i \Sigma \rightarrow \psi \right) \right),$$

from which we easily get

$$\Omega_2 \vdash \forall x : \mathbf{set} \left(\text{Tr}(x) \wedge \text{Axiom}^2(\gamma) \rightarrow x \subseteq \text{ST}_C \left(\bigwedge_{i=0}^k \Box^i \Sigma \rightarrow \psi \right) \right).$$

Let $\Sigma = \{\sigma_1, \dots, \sigma_m\}$. Then

$$\text{ST}_C \left(\bigwedge_{i=0}^k \Box^i \Sigma \rightarrow \psi \right) = (x \setminus \bigcap_{i=0}^k \bigcap_{j=1}^m \text{Pow}^i(\text{ST}_C(\sigma_j))) \cup \text{ST}_C(\psi).$$

We are now going to prove that, if $\sigma \in \Sigma$, then

$$\Omega_2 \vdash \forall x : \mathbf{set}(\bigwedge_{i=1}^n \text{ST}^2(\varphi_i) \rightarrow x \subseteq \text{ST}_C(\sigma)). \quad (1)$$

If $\sigma \in \Sigma$, then $\vdash \sigma\{\vec{Q}/\vec{\varphi}\}$. From this we get

$$\vdash \forall y : \mathbf{set} \sigma\{\vec{Q}/\vec{\varphi}_i(\vec{y})\}.$$

Next one has to prove that

$$\Omega_2 \vdash \forall x : \mathbf{set}(\bigwedge_{i=0}^n \text{ST}^2(\varphi_i) \rightarrow \forall y : \mathbf{set}(y \in x \rightarrow (y \in \text{ST}_C(\sigma) \leftrightarrow \sigma\{\vec{Q}/\vec{\varphi}_i(\vec{y})\}))),$$

Which is done by a straightforward structural induction on σ and we omit it.

In particular, if $\sigma \in \Sigma$, $\vdash \forall y : \mathbf{set} \sigma\{\vec{Q}/\vec{\varphi}_i(\vec{y})\}$. This, together with what has just been proved, gives

$$\Omega_2 \vdash \forall x : \mathbf{set}(\bigwedge_{i=1}^n \text{ST}^2(\varphi_i) \rightarrow \forall y : \mathbf{set}(y \in x \rightarrow y \in \text{ST}_C(\sigma))),$$

that is (1).

Now it is easy to show, by induction on i , that, for $i \in \mathbb{N}$ and $\sigma \in \Sigma$,

$$\Omega_2 \vdash \forall x : \mathbf{set}(\text{Tr}(x) \wedge \bigwedge_{i=1}^n \text{ST}^2(\varphi_i) \rightarrow x \subseteq \text{Pow}^i(\text{ST}_C(\sigma))).$$

The case $i = 0$ is the previous result.

Assume the thesis holds for i and let x be an element of sort $\mathbf{set}$ such that $\text{Tr}(x) \wedge \bigwedge_{i=1}^n \text{ST}^2(\varphi_i)$. Then by induction hypothesis $x \subseteq \text{Pow}^i(\text{ST}_C(\sigma))$, by monotonicity of Pow (which easily follows from the axioms) $\text{Pow}(x) \subseteq \text{Pow}(\text{Pow}^i(\text{ST}_C(\sigma))) = \text{Pow}^{i+1}(\text{ST}_C(\sigma))$, by $\text{Tr}(x)$ we get $x \subseteq \text{Pow}(x)$, and hence $x \subseteq \text{Pow}^{i+1}(\text{ST}_C(\sigma))$.

Therefore the following are theorems of Ω_2 :

$$\forall x : \mathbf{set}(\text{Tr}(x) \wedge \bigwedge_{i=1}^n \text{ST}^2(\varphi_i) \rightarrow x \subseteq \bigcap_{i=0}^k \bigcap_{j=1}^m \text{Pow}^i(\text{ST}_C(\sigma_j))),$$

$$\forall x : \mathbf{set}(\text{Tr}(x) \wedge \bigwedge_{i=1}^n \text{ST}^2(\varphi_i) \rightarrow \forall y : \mathbf{set}(y \in z \cup \text{ST}_C(\psi) \leftrightarrow y \in \text{ST}_C(\psi))),$$

$$\text{with } z = x \setminus \bigcap_{i=0}^k \bigcap_{j=1}^m \text{Pow}^i(\text{ST}_C(\sigma_j)),$$

$$\forall x : \mathbf{set}(\text{Tr}(x) \wedge \bigwedge_{i=1}^n \text{ST}^2(\varphi_i) \rightarrow \forall y : \mathbf{set}(y \in \text{ST}_C\left(\bigwedge_{i=0}^k \square^i \Sigma \rightarrow \psi\right) \leftrightarrow y \in \text{ST}_C(\psi))).$$

From this we have that $\Omega_2 \vdash \forall x : \mathbf{set}(\text{Tr}(x) \wedge \text{Axiom}^2(\gamma) \wedge \bigwedge_{i=1}^n \text{ST}^2(\varphi_i) \rightarrow x \subseteq \text{ST}_C(\psi))$, that is, $\Omega_2 \vdash (\gamma, F)^{*2}$. ■

Theorem 11 [*Soundness of the translation*] $\Omega_2 \vdash (\gamma, F)^{*2} \implies \models_{\mathbf{L}} F$.

Proof. Let $\mathcal{A}$ be a **Q1-L**-model such that $\mathcal{A} \models \{\vec{Q} \leftrightarrow \vec{\varphi}\}$. Build $\mathcal{U} \setminus V_{\alpha+1}$ starting from the propositional part of $\mathcal{A}$ as was done in the proof of soundness for the propositional $\Box$ -as-*Pow*, see [4]. We want to extend this to a model $\mathfrak{B}$ for the language L' in such a way that $\mathfrak{B}, [x/\mathcal{W}^*] \models \bigwedge_{i=1}^n \text{ST}^2(\varphi_i)$.

For the sort **term** we use the same domain, constant and function symbol interpretation as in $\mathcal{A}$. If p is an n -ary relation symbol of L , let $p^{\mathfrak{B}}(a_1, \dots, a_n, b) \iff b \in \mathcal{W}^*$ (and hence $b = w^*$ for some $w \in \mathcal{W}$) and $w \models p(a_1, \dots, a_n)$.

Now we prove, by structural induction on φ , that for every classical first-order formula φ , assignment s and world $w \in \mathcal{W}$, $\mathfrak{B}, s[y/w^*] \models \bar{\varphi}(y) \iff w, s \models \varphi$.

If $\varphi = p(t_1, \dots, t_n)$, the thesis holds by definition.

If $\varphi = \neg\theta$, then $\mathfrak{B}, s[y/w^*] \models \bar{\varphi}(y) \iff \mathfrak{B}, s[y/w^*] \not\models \bar{\theta}(y) \iff$ (by inductive hypothesis) $w, s \not\models \theta \iff w, s \models \varphi$.

If $\varphi = \theta \vee \eta$, then $\mathfrak{B}, s[y/w^*] \models \bar{\varphi}(y) \iff \mathfrak{B}, s[y/w^*] \models \bar{\theta}(y)$ or $\mathfrak{B}, s[y/w^*] \models \bar{\eta}(y) \iff$ (by inductive hypothesis) $w, s \models \theta$ or $w, s \models \eta \iff w, s \models \varphi$.

If $\varphi = \forall y \theta$, then $\mathfrak{B}, s[y/w^*] \models \bar{\varphi}(x) \iff$ for all a of sort **term** $\mathfrak{B}, s[y/w^*][z/a] \models \bar{\theta}(y) \iff$ for all a of sort **term** $\mathfrak{B}, s[z/a][y/w^*] \models \bar{\theta}(y) \iff$ (by inductive hypothesis) for all a of sort **term** $w, s[z/a] \models \theta \iff w, s \models \varphi$.

Thus, if φ is closed, $\mathfrak{B}, [y/w^*] \models \bar{\varphi}(y)$ if and only if $w \models \varphi$ in $\mathcal{A}$.

We must still give an interpretation to the constant symbols $c_1 \dots c_n$. Let $c_i^{\mathfrak{B}} = Q_i^* = \{w^* \mid w \models Q_i\} = \{w^* \mid w \models \varphi_i\}$. It is easy to see that, under this interpretation, $\mathfrak{B}, [x/\mathcal{W}^*] \models \bigwedge_{i=1}^n \text{ST}^2(\varphi_i)$ holds. Since also $\mathfrak{B}, [x/\mathcal{W}^*] \models \text{Tr}(x)$ and $\mathfrak{B}, [x/\mathcal{W}^*] \models \text{Axiom}^2(\gamma)$ hold, from the hypothesis $\Omega_2 \models (\gamma, F)^{*2}$ it follows that $\mathfrak{B}, [x/\mathcal{W}^*] \models x \subseteq \text{ST}_C(\psi)$. From Lemma 8 of [4] it follows now that $\mathcal{A} \models \psi$. ■

6 One-Sorted Translation of Locally Quantified Formulas

All the results of the previous section hold also for the one-sorted case, in which we simply *strip* all the formulas of the sort denotations. The resulting translation is called $(\gamma, F)^{*1}$. The proofs are very similar, but slightly more complicated, thus we preferred to introduce the two-sorted version first.

Theorem 12 $\vdash_{\gamma} F \iff \Omega \vdash (\gamma, F)^{*1}$.

With respect to the two-sorted case, the proof changes in the soundness part only, where it is now necessary to embed the domain of quantification into the universe of sets. The proof follows the lines of that of the two-sorted case in a quite straightforward manner, but jumping forward and back through embeddings and inverse embeddings several times.

Conclusions

On the basis of a separation result that allows us to distinguish (classical) first-order from modal (propositional) reasoning, we showed how the $\Box$ -as-*Pow* translation can be generalized to deal with **Q1** locally quantified modal formulas. In [10], we proved that the $\Box$ -as-*Pow* translation can actually be extended to the whole **Q1** modal system. If one tries to apply the ideas presented in this paper to translate non-rigid modal logics, some form of dependency between the interpretation of the symbols and the world should be introduced. We are currently investigating the possibility of extending our technique by introducing an extra argument (the world) to the translation function.

References

1. P. Aczel. *Non-Well-Founded Sets*. Number 14 in CSLI Lecture Notes. CSLI, Stanford, California, 1988.
2. J. van Benthem, G. D'Agostino, A. Montanari, and A. Policriti. Modal deduction in second-order logic and set theory — I. *Journal of Logic and Computation*, 7(2):251–265, 1997.
3. J. van Benthem, G. D'Agostino, A. Montanari, and A. Policriti. Modal deduction in second-order logic and set theory — II. *Studia Logica*, 60(3):387–420, 1998.
4. G. D'Agostino, A. Montanari, and A. Policriti. A set-theoretic translation method for polymodal logics. *Journal of Automated Reasoning*, 15(3):314–337, 1995.
5. M. Fitting. Basic modal logic. In D. M. Gabbay, C. J. Hogger, and J. A. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume I, pages 395–448. Oxford University Press, Oxford, 1993.
6. D. M. Gabbay and H. J. Ohlbach. Quantifier elimination in second-order predicate logic. *South African Computer Journal*, 7:35–43, 1992.
7. J. W. Garson. Quantification in modal logic. In D. M. Gabbay and F. Guenther, editors, *Handbook of Philosophical Logic*, volume II, pages 249–307. Reidel, Dordrecht, The Netherlands, 1984.
8. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, Berlin, 1995.
9. A. Montanari and A. Policriti. A set-theoretic approach to automated deduction in graded modal logics. In M. E. Pollack, editor, *Proceedings of the 15th IJCAI*, pages 196–201. Morgan Kaufmann, 1997.
10. A. Montanari, A. Policriti, and Slanina M. Supporting automated deduction in first-order modal logics. In A. G. Cohn, F. Giunchiglia, and B. Selman, editors, *Proceedings of the 7th KR*, pages 547–556. Morgan Kaufmann, 2000.
11. A. Nonnengart. First-order modal logic theorem proving and functional simulation. In R. Bajcsy, editor, *Proceedings of the 13th IJCAI*, pages 80–85. Morgan Kaufmann, 1993.
12. H. J. Ohlbach. Translation methods for non-classical logics — An overview. *Bulletin of the IGPL*, 1(1):69–89, 1993.
13. H. J. Ohlbach. Combining hilbert style and semantic reasoning in a resolution framework. In C. Kirchner and H. Kirchner, editors, *Proceedings of CADE-15*, volume 1421 of *Lecture Notes in Artificial Intelligence*, pages 205–219. Springer, 1998.

π -Calculus, Structured Coalgebras, and Minimal HD-Automata^{*}

Ugo Montanari¹ and Marco Pistore²

¹ Computer Science Department, Corso Italia 40, 56100 Pisa, Italy

² ITC-IRST, Via Sommarive 18, 38050 Povo (Trento), Italy

Abstract. The coalgebraic framework developed for the classical process algebras, and in particular its advantages concerning minimal realizations, does not fully apply to the π -calculus, due to the constraints on the freshly generated names that appear in the bisimulation.

In this paper we propose to model the transition system of the π -calculus as a coalgebra on a category of name permutation algebras and to define its abstract semantics as the final coalgebra of such a category. We show that permutations are sufficient to represent in an explicit way fresh name generation, thus allowing for the definition of minimal realizations.

We also link the coalgebraic semantics with a slightly improved version of *history dependent* (HD) automata, a model developed for verification purposes, where states have local names and transitions are decorated with names and name relations. HD-automata associated with agents with a bounded number of threads in their derivatives are finite and can be actually minimized. We show that the bisimulation relation in the coalgebraic context corresponds to the minimal HD-automaton.

1 Introduction

The π -calculus [8] is probably the best studied calculus for name mobility, and the basis for several proposals concerning higher order mobility, security and object orientation. Also, π -calculus expressiveness can be considered the touchstone for a number of formalisms exploring the needs of wide area programming. The advantage of the π -calculus is its simplicity and its process algebra flavor, e.g., its operational semantics given by means of a transition system and its abstract semantics based on bisimilarity.

However, while a process calculus like CCS, at least in the strong case, can be easily casted in a coalgebraic framework [12], the π -calculus requires some care. In fact, consider the definition of early bisimulation:

Definition 1. A relation $\mathcal{R}$ over agents is an early simulation if $P \mathcal{R} Q$ implies:

- for each $P \xrightarrow{\alpha} P'$ with $\text{bn}(\alpha) \cap \text{fn}(P, Q) = \emptyset$ there is some $Q \xrightarrow{\alpha} Q'$ such that $P' \mathcal{R} Q'$.

^{*} Research supported by CNR Integrated Project *Sistemi Eterogenei Connessi mediante Reti*; by Esprit Working Groups *CONFER2* and *COORDINA*; and by MURST project *TOSCA*.

A relation $\mathcal{R}$ is an early bisimulation if both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are early simulations. Early bisimilarity $\sim_\pi$ is the largest early bisimulation.

Notice the condition $\text{bn}(\alpha) \cap \text{fn}(P, Q) = \emptyset$: the first agent is not allowed to create new names that are already syntactically present in the second agent. Thus the bisimilarity class of an agent cannot be defined “in isolation”, but only relatively to possible partners, or at least to their free names. As a consequence, the coalgebraic framework does not fully apply. In practice, algorithms for checking bisimilarity based on the above definition are only of the “on the fly” kind, and with them it is not possible to construct the minimal equivalent agent. To apply the standard definition of bisimulation also in this context, it is necessary to define a mechanism of name allocation which guarantees that the extruded names are chosen in a consistent way by the agents P and Q . The choice of this mechanism is critical, as it decides whether the obtained transition systems are finite- or infinite-states: to obtain finite-state models, in fact, it is necessary not only to define how fresh names are allocated, but also how unused names are deallocated, so that they can be used again.

A coalgebraic semantics for various versions of the π -calculus has been proposed in [7], but the approach is higher order, and it is not amenable to finite state verification. Moreover, the semantics is still parametrized by the set of names of the possible partners. Since this set of names continues to grow during the evolution of an agent, this semantics generates infinite-state models.

In the paper we propose a standard coalgebraic definition of the π -calculus semantics, which is based on name permutations. The effect of permutations on the behavior of agents is, in our opinion, the smallest information required to define a semantically correct mechanism of name deallocation. In fact, as we will see, according to our approach the permutations associated with an agent in the final coalgebra define which names are really “active” in the agent, and equivalent agents have the same active names.

We also link our coalgebraic semantics to a slightly improved version of *history dependent* (HD) automata [9,11], a model for named calculi, where the states are enriched with local names and the transitions are decorated with names and name relations. HD-automata allow for name deallocation and are hence suitable for verification purposes [4,5]. We show that the bisimilarity relation corresponds to the minimal automaton. Furthermore, automata associated with agents with a bounded number of threads in their derivatives are finite and can be actually minimized.

The full version of this paper [10] contains all the explanations and the proofs that are omitted here for lack of space.

2 Names and Permutations

We denote with $\mathfrak{N} = \{x_0, x_1, x_2, \dots\}$ the infinite, countable set of *names* and we use $x, y, z, \dots$ to denote names. A *name substitution* is a function $\sigma : \mathfrak{N} \rightarrow \mathfrak{N}$. We denote with $\sigma \circ \sigma'$ the composition of substitutions σ and σ' ; that is,

$\sigma \circ \sigma'(x) = \sigma(\sigma'(x))$. A *name permutation* is a bijective name substitution. We use ρ to denote a permutation.

The *kernel* $K(\rho)$ of a permutation ρ is the set of the names that are changed by the permutation: $K(\rho) = \{n \mid \rho(n) \neq n\}$. We say that permutation ρ has *finite kernel* if set $K(\rho)$ is finite. A finite-kernel permutation leaves unchanged all but a finite subset of the names.

A *symmetry* S for $\mathfrak{N}$ is a group of finite-kernel permutations. That is, S is a set of finite-kernel permutations such that whenever $\rho, \rho' \in S$, then also $\rho \circ \rho' \in S$. We denote with Sym the set of the symmetries of $\mathfrak{N}$, and we use S to denote a symmetry. A set G of permutations is a *lateral class* of symmetry S if there is a (not necessarily finite-kernel) permutation ρ such that $G = \{\rho \circ \rho' \mid \rho' \in S\}$. We denote with $Lat(S)$ the set of the lateral classes of symmetry S and with Lat the set of all the lateral classes.

3 A Permutation Algebra for the π -Calculus

The algebraic specification that we use in this paper is very simple: it contains only unary operations that correspond to name permutations. As we have explained in the introduction, the effects of permutations on agents give the smallest information required to define name deallocation.

A permutation algebra is defined by a carrier set, corresponding to the states, and by a description of how states are transformed by the (finite-kernel) permutations. The permutation algebra corresponding to the π -calculus agents is defined by taking the agents as the carrier set and by interpreting the permutations as name substitutions.

Definition 2 (permutation algebras). *Algebraic specification $\Gamma_\pi = \langle \Sigma_\pi, E_\pi \rangle$ is defined as follows:*

- Σ_π is the set of finite-kernel permutations of $\mathfrak{N}$; they are all unary operations;
- E_π are the classical axioms of permutation, namely:

$$(\rho \circ \rho')(X) = \rho(\rho'(X)) \quad \text{and} \quad \text{id}(X) = X.$$

A permutation algebra is an algebra for specification Γ_π .

The permutation algebra A_π for the π -calculus is defined as follows:

- $|A_\pi|$ are the π -calculus agents up to structural congruence;
- $\rho^A(P) = P\rho$.

Definition 3 (orbit, symmetry and support). *Let A be a permutation algebra and let X be an element of the carrier set of A .*

The orbit of X is the set of states $\text{orbit}^A(X) = \{X' \mid X' = \rho^A(X) \text{ for some } \rho\}$. The symmetry of X is the group of the permutations that map X into itself: $\text{sym}^A(X) = \{\rho \mid X = \rho^A(X)\}$.

The support of X is the smallest set of names $\text{supp}^A(X)$ such that, given a permutation ρ , if $\rho(n) = n$ for all $n \in \text{supp}^A(X)$, then $\rho^A(X) = X$.

We will often omit the superscript A in $\text{orbit}^A(X)$, $\text{sym}^A(X)$ and $\text{supp}^A(X)$, whenever the algebra we refer to is clear from the context.

In the case of A_π , orbits correspond to sets of agents that differ for name permutations. The support of an agent P corresponds to the set $\text{fn}(P)$ of its free names: in fact, any permutation that does not affect $\text{fn}(P)$ maps P into itself. So, $\text{sym}(P)$ contains all the permutations that do not affect $\text{fn}(P)$. Notice however that, since we consider π -calculus agents up to structural congruence, then $\text{sym}(P)$ may contain also non-trivial permutations that affect names in $\text{fn}(P)$. For instance, since $\bar{x}x.\mathbf{0} + \bar{y}y.\mathbf{0} \equiv \bar{y}y.\mathbf{0} + \bar{x}x.\mathbf{0}$, then the permutation that exchanges x and y is in $\text{sym}(\bar{x}x.\mathbf{0} + \bar{y}y.\mathbf{0})$.

Orbits partition the set of states (i.e., the carrier) of a permutation algebra in disjoint blocks: in fact, $\text{orbit}(X') = \text{orbit}(X)$ if and only if $X' \in \text{orbit}(X)$. The following result shows that the behavior of all the elements of an orbit can be predicted giving the behavior of any single element of the orbit: in fact, we can give a compact version of a substitution algebra, that is based on representing the elements of an orbit as pairs consisting of a *canonical representative* of the orbit and of the lateral classes of its symmetry.

Definition 4 (canonical representatives of orbits). *Let o be an orbit. We assume to have a canonical representative of o , that we denote by $\text{cr}(o)$.*

Proposition 1. *Permutation algebra A is isomorphic to A_{cr} , where:*

- $|A_{cr}| = \{ \langle \text{cr}(\text{orbit}(X)), G \rangle \mid X \in |A|, G \in \mathcal{Lat}(\text{sym}(\text{cr}(\text{orbit}(X)))) \}$; and
- $\rho^{A_{cr}}(\langle cr, G \rangle) = \langle cr, \{ \rho \circ \rho' \mid \rho' \in G \} \rangle$.

4 A Permutation Coalgebra for the π -Calculus

In the previous section we have defined the permutation algebra that we use to model π -calculus agents. Here we specify the coalgebra corresponding to its (early) operational semantics. By exploiting the approach of *structured coalgebras* [2,3], the coalgebraic operational semantics is built on the top of the permutation algebra: therefore, the effects of permutations on the evolutions of the agents are taken explicitly into account.

The standard operational semantics of the π -calculus is defined via labelled transitions $P \xrightarrow{\alpha} P'$, where P is the starting agent, P' is the target one and α is an action. There are four kinds of actions: synchronizations τ , inputs xy , free outputs $\bar{x}y$, and bound outputs $\bar{x}(z)$; in the case of the bound output action $\alpha = \bar{x}(z)$ we say that z is a bound name ($z \in \text{bn}(\alpha)$). We refer to [8] for the definition of the π -calculus transition relation. Here we comment only on the bound output transitions: a bound output corresponds to the emission of a private name of an agent to the environment: in this way, the channel becomes public and can be used for further communications between the agent and the environment.

The peculiarity of the coalgebraic operational semantics that we are going to define, if compared to the standard π -calculus operational semantics, is the way

bound output transitions are represented. In the standard π -calculus semantics, the creation of the new channel is modeled by picking any fresh name to represent the new channel. Here, instead, we follow a different approach: in the target state, name x_0 is used to denote the newly created name, and, to avoid any name clashing, during a bound output transition names x_i present in the source state are renamed to x_{i+1} . That is, the creation of a new name is modeled by shifting all the preexisting names and by using x_0 to represent the new name.

First of all we define the set L_π of the labels:

$$L_\pi = \{\text{tau}, \text{in}(x, y), \text{out}(x, y), \text{bout}(x) \mid x, y \in \mathfrak{N}\}. \quad (1)$$

If $l \in L_\pi$ then $\sigma(l)$ is the label obtained from l by applying substitution σ .

The correspondence between labels L_π and the actions of the π -calculus is the obvious one, except for the bound output transitions, where only the channel x on which the output occurs is observed in label $\text{bout}(x)$. In fact, we know that the extruded channel corresponds to name x_0 in the target state.

Now we are ready to define the transition specification for the π -calculus. We recall from [3] that a *transition specification* $\Delta = \langle \Gamma, L, R \rangle$ consists of an algebraic specification Γ , that defines the algebra of the states, of a set of labels L for the transitions, and of a set R of rules that the transition systems should satisfy. A transition system *lts* for specification Δ is defined by an algebra A on specification Γ and by a set of L -labelled transitions $\Rightarrow$ that respect rules R .

The only rules that appear in the specification that we are going to define are those that describe the effect of the permutations on the transition relation; they are straightforward, except for the bound output transitions. In this case, in fact permutation ρ transforms transition $P \xRightarrow{\text{bout}(x)} P'$ into $\rho(P) \xRightarrow{\text{bout}(\rho(x))} \rho_{+1}(P')$, where permutation ρ_{+1} is obtained by shifting permutation ρ to the right:

$$\frac{-}{\rho_{+1}(x_0) = x_0} \quad \frac{\rho(x_n) = x_m}{\rho_{+1}(x_{n+1}) = x_{m+1}} \quad (2)$$

Definition 5 (transition specification Δ_π). *The transition specification Δ_π for the π -calculus is the tuple $\langle \Gamma_\pi, L_\pi, R_\pi \rangle$, where the algebraic specification Γ_π is as in Definition 2, labels L_π are defined in Equation 1 and the rules are the following ones, where ρ_{+1} is defined as in Equation 2:*

$$\frac{X \xRightarrow{l} X'}{\rho(X) \xRightarrow{\rho(l)} \rho(X')} \text{ for } l = \text{tau}, \text{in}(x, y), \text{out}(x, y) \quad \frac{X \xRightarrow{\text{bout}(x)} X'}{\rho(X) \xRightarrow{\text{bout}(\rho(x))} \rho_{+1}(X')}$$

We denote with LTS_π the class of transition systems that are models of Δ_π .

We remark that specification Δ_π does not contain any axiom; therefore, the initial model for the specification is empty. This is correct, as our specification is not intended to define a particular transition system; rather, it only defines the expected effects of permutations on the transitions.

To define the transition system of LTS_π corresponding to the operational semantics of the π -calculus, particular care is necessary for the bound output transitions, due to the particular way that we use to represent the creation of new channels, as explained at the beginning of this section. More precisely, for each π -calculus transition $P \xrightarrow{\bar{x}(y)} P'$ we generate a transition $P \xRightarrow{\text{bout}(x)}_\pi P''$, where agent P'' is obtained from P' by shifting all its names and by mapping y into x_0 . That is, $P'' = P'\sigma$, where:

$$\sigma(y) = x_0 \quad \sigma(x_n) = x_{n+1} \text{ if } x_n \neq y. \quad (3)$$

Definition 6 (transition system LTS_π). *The transition system corresponding to the early operational semantics of the π -calculus is $\text{LTS}_\pi = \langle A_\pi, \Rightarrow_\pi \rangle$, where A_π is the permutation algebra corresponding to the π -calculus agents, and $\Rightarrow_\pi$ is defined by the following axioms, with σ defined as in Equation 3:*

$$\frac{P \xrightarrow{\tau} P'}{P \xRightarrow{\text{tau}}_\pi P'} \quad \frac{P \xrightarrow{xy} P'}{P \xRightarrow{\text{in}(x,y)}_\pi P'} \quad \frac{P \xrightarrow{\bar{x}y} P'}{P \xRightarrow{\text{out}(x,y)}_\pi P'} \quad \frac{P \xrightarrow{\bar{x}(y)} P'}{P \xRightarrow{\text{bout}(x)}_\pi P'\sigma}$$

To prove that LTS_π is really an element of LTS_π , we should show that it satisfies the permutation rules given in Definition 5.

Proposition 2. *Let $P \xRightarrow{l}_\pi Q$ be a transition of LTS_π . If $P' = \rho(P)$, then $P' \xRightarrow{l'}_\pi Q'$, where $l' = \rho(l)$ and $Q' = \rho(Q)$ (resp. $Q' = \rho_{+1}(Q)$ if $l = \text{bout}(x)$).*

A first important result is that the standard bisimilarity relation on LTS_π coincides with the early π -calculus bisimilarity (defined in the introduction).

Theorem 1. *Let $\sim_{\text{LTS}_\pi}$ be the bisimilarity relation on the states of LTS_π . Then $P \sim_{\text{LTS}_\pi} Q$ iff $P \sim_\pi Q$.*

Other results on LTS_π and in particular on LTS_π are obtained by applying the theory of [3]. There, sufficient conditions on a transition specification $\Delta = \langle \Gamma, L, R \rangle$ and on a particular transition system LTS are spelled out for the following results to hold: (i) the powerset functor P_L corresponding to the ordinary interpretation of labelled transition systems [12] can be lifted from the category **Set** to the functor P_L^Δ on the category **Alg**(Γ) of the algebras satisfying specification Γ ; this yields a category $P_L^\Delta\text{-Coalg}$. (ii) $P_L^\Delta\text{-Coalg}$ is equipped with a final object, and the unique morphism to it from any coalgebra in $P_L^\Delta\text{-Coalg}$ defines on the algebra of states a relation which is both a congruence and the maximal bisimulation (bisimilarity); the labelled transition system LTS_π is an object of $P_L^\Delta\text{-Coalg}$ (simply **Coalg** $_\pi$), and the unique morphism from it to the final coalgebra (which can also be seen as a transition system with operations) defines $\sim_{\text{LTS}_\pi}$.

The condition on the transition specification requires that the rules and the axioms respect particular formats (for the rules an extension of *De Simone* format). This condition is fulfilled by the specification of Definition 5. For the particular transition system, a *homomorphism* condition must be satisfied, which

requires that, whenever the transition $op(a_1, \dots, a_n) \xRightarrow{l} b$ is present, then it must be derivable using the rules of the transition specification, for all the decompositions $op(a_1, \dots, a_n)$ of the source which hold in the algebra of states. This condition is satisfied by lts_π (actually, by all the transition systems in LTS_π) thus allowing for the application of the results described above.

Theorem 2. *Let $lts \in LTS_\pi$. Then transition system lts satisfies the homomorphism property. In particular, lts_π satisfies the homomorphism property.*

Therefore, bisimilarity $\sim_{lts_\pi}$ is a congruence w.r.t. the operators of algebra A_π : i.e., $\sim_\pi$ is closed for permutations.

Moreover, bisimilarity $\sim_{lts_\pi}$ is induced by the unique morphism from lts_π to the final object in $\mathbf{Coalg}_\pi$.

We have stated the results above only in the case of the π -calculus *early* operational semantics. However, these results can be easily applied to other “dialects” of the π -calculus, as the *late* operational semantics. In particular, Proposition 2 defines the requirement for a transition system to be mapped in the coalgebraic framework: namely, it must satisfy the rules of Definition 5.

Notice that the states of the final model in $\mathbf{Coalg}_\pi$ form a Γ_π -algebra. So, in particular, a support $\text{supp}(X)$ is defined for each of these states. The support of a π -calculus agent in lts_π defines the free names of the agent; the support of the corresponding state in the final model, instead, define the “active names” [9, 11] of the agent, i.e., those names that play a role in the evolution of the agent. While equivalent π -calculus agents may have different free names, they have the same active names (as they correspond to the same state in the final model). We remark that the support of a state is defined from the effects of the permutations on the state (see Definition 3); therefore it is clear that permutations are the basic ingredient, at least in the coalgebraic framework, to define active names and, hence, to allow for the deallocation of unused names.

5 HD-Automata

In this section we show how HD-automata can be used to give a compact representation of lts_π . HD-automata [9,11] are an operational model introduced by the authors to give compact representations of the behavior of concurrent calculi with names. Their most interesting feature is that they allow to represent the behavior of these systems up to name permutations. In fact, each state of the automaton is a concise representation of an orbit of lts_π , and transitions between pairs of HD-states represent all the transitions between the corresponding orbits. This is possible since every π -calculus agent has finite support, and thus its orbit contains only a finite amount of information. This compact representation is obtained as the names that appear in a state of a HD-automaton do not have a “global” identity: they only have an identity that is “local” to the state; whenever a transition is performed, a name correspondence is explicitly required to describe how the “local” names of the source state are related to the “local” names of the target.

Now we define a variant of HD-automata with Symmetries that is adequate for the early semantics of the π -calculus. We refer to [11] for a complete presentation of the theory of the different classes of HD-automata.

Definition 7 (HD-automata). A HD-automaton with Symmetries (or simply HD-automaton) $\mathcal{A}$ is a tuple $\langle \mathcal{Q}, \text{sym}, L_\pi, \mapsto \rangle$, where:

- $\mathcal{Q}$ is the set of states;
- $\text{sym} : \mathcal{Q} \rightarrow \text{Sym}$ associates to each state its symmetry;
- labels L_π are defined as in Equation 1;
- $\mapsto \subseteq \{\langle Q, l, \rho, Q' \rangle \mid Q, Q' \in \mathcal{Q}, l \in L_\pi, \rho \text{ is a name permutation}\}$ is the transition relation, where Q and Q' are, respectively, the starting and the target state, l is the label, and ρ is a permutation, that describes how the names of the target state Q' correspond, along this transition, to the names of the source state Q . Whenever $\langle Q, l, \rho, Q' \rangle \in \mapsto$ then we write $Q \xrightarrow{l}_\rho Q'$.

Now we define HD-bisimulation, following the approach in [9,11]. Since the names that appear in the states of the HD-automata have only a local identity, a HD-bisimulation is defined as a set of triples $\langle Q_1, \delta, Q_2 \rangle$, where δ is a (not necessarily finite-kernel) permutation of $\mathfrak{N}$ that sets a correspondence between the names of Q_1 and those of Q_2 .

Definition 8 (HD-bisimulation). Let $\mathcal{A}$ be a HD-automaton. A HD-simulation for $\mathcal{A}$ is a set of triples $\mathcal{R} \subseteq \{\langle Q_1, \delta, Q_2 \rangle \mid Q_1, Q_2 \in \mathcal{Q}, \delta : \mathfrak{N} \leftrightarrow \mathfrak{N}\}$ such that, whenever $\langle Q_1, \delta, Q_2 \rangle \in \mathcal{R}$ then:

- for each $\rho_1 \in \text{sym}(Q_1)$ and each $Q_1 \xrightarrow{l_1}_{\sigma_1} Q'_1$, there exist some $\rho_2 \in \text{sym}(Q_2)$ and some $Q_2 \xrightarrow{l_2}_{\sigma_2} Q'_2$, such that:
 - $l_2 = \gamma(l_1)$, where $\gamma = \rho_2^{-1} \circ \delta \circ \rho_1$;
 - $\langle Q'_1, \delta', Q'_2 \rangle \in \mathcal{R}$, where: $\delta' = \begin{cases} \sigma_2^{-1} \circ \gamma \circ \sigma_1 & \text{if } l_1 \neq \text{bout}(x) \\ \sigma_2^{-1} \circ \gamma_{+1} \circ \sigma_1 & \text{if } l_1 = \text{bout}(x). \end{cases}$

A HD-bisimulation for $\mathcal{A}$ is a set of triples $\mathcal{R}$ such that both $\mathcal{R}$ and $\mathcal{R}^{-1} = \{\langle Q_2, \delta^{-1}, Q_1 \rangle \mid \langle Q_1, \delta, Q_2 \rangle \in \mathcal{R}\}$ are HD-simulations for $\mathcal{A}$.

According to this definition, each transition from state Q_1 is considered many times, once for every permutation ρ_1 in $\text{sym}(Q_1)$: in HD-automata a single transition is used to represent a whole set of transitions that differ for a permutation that belongs to the symmetry of the source state. Moreover, two transitions $Q_1 \xrightarrow{l_1}_{\sigma_1} Q'_1$ and $Q_2 \xrightarrow{l_2}_{\sigma_2} Q'_2$ match only if they have the same label (up to the appropriate permutation γ), and the target states are related in the HD-bisimulation, via a correspondence δ' that is obtained from γ by applying the names correspondences σ_1 and σ_2 . In the case of bound output transitions, permutation γ is shifted before applying substitutions σ_1 and σ_2 : this is necessary to take into account the generation of a new name performed during the transition.

Now we show that HD-automata allow for a compact representation of the transition systems in LTS_π : the following definition shows how to build a HD-automaton by taking a state for each orbit of the transition system.

Definition 9 (from transition systems to HD-automata).

Let $lts = \langle A, \Longrightarrow \rangle$ be any transition system of LTS_π . Then the corresponding HD-automaton $\mathcal{A} = \langle \mathcal{Q}, \text{sym}, L_\pi, \mapsto \rangle$ is defined as follows:

- $\mathcal{Q} = \{\text{cr}(\text{orbit}(Q)) \mid Q \in |A|\};$
- $\text{sym}(Q) = \text{sym}^A(Q);$
- $Q \mapsto_\sigma Q'$ if $Q \xRightarrow{l} Q'', Q' = \text{cr}(\text{orbit}(Q''))$ and $\sigma(Q') = Q''$.

Theorem 3. Let lts be a transition system in $\mathbf{Coalg}_\pi$ and let $\mathcal{A}$ be the corresponding HD-automaton according to Definition 9. To every morphism $m : lts \rightarrow lts'$ in $\mathbf{Coalg}_\pi$ corresponds a HD-bisimulation on $\mathcal{A}$.

Let lts_f be the final element of $\mathbf{Coalg}_\pi$ and let f be the unique morphism $f : lts \rightarrow lts_f$. Then the HD-bisimulation corresponding to morphism f is the largest HD-bisimulation for $\mathcal{A}$. Finally, the HD-automaton corresponding to the image in lts_f of lts is the minimal HD-automaton equivalent to $\mathcal{A}$.

6 Concluding Remarks

The coalgebraic semantics of the π -calculus presented in this paper allows for the definition of the “minimal” transition system for a given π -calculus agent: it is sufficient to get in the final object of the category $\mathbf{Coalg}_\pi$ the image of the transition system corresponding to the agent. In [11,10] it is shown that HD-automata allow for a more explicit definition of minimal realizations: the minimal HD-automaton corresponding to HD-automaton $\mathcal{A}$ is obtained by quotienting $\mathcal{A}$ with respect to the largest HD-bisimulation for $\mathcal{A}$. When π -calculus agents with a bounded number of threads in their derivatives are considered, the relevant parts of the HD-automaton are finite. Therefore, the minimal HD-automata can be effectively constructed for these agents.

The algebraic structure we considered is as reduced as possible. We think that interesting lines of development would result in extending the structure to some of the syntactic operators of the π -calculus, like restriction and parallel composition. However it will not be possible to include all the π -calculus constructs, since it is known that the abstract semantics of the π -calculus is not a congruence for input prefixes, while such a property would be automatically guaranteed by the structured coalgebra framework.

The approach presented in this paper has some analogies with two papers on domain equations for the π -calculus presented at LICS'96 [6,13]. The latter work is based on the category of covariant (pullback preserving) presheaves over the category I of names, and on a functor on presheaves defined using exponentiation to model input and a “differentiation” functor to model bound output. While our category of permutation algebras could also be seen as a functor category in the Lawvere style, our approach looks rather different. In fact we do not give a denotational semantics, being interested in a “flat” version of the π -calculus where the only operations are name permutations, and, technically, our approach is based on the structured coalgebra results [3]. In addition, our

construction is first order, since input and free output are modeled in a similar way, and permutations across bound output transitions are defined using the “ ρ_{+1} ” operation on permutations. However, a full comparison between the two approaches deserves further study.

Another work related to ours is described in [1]. There, final coalgebras are used to define a semantic model for the π -calculus that is fully abstract for early bisimulation and that allows for a compositional interpretation of the π -calculus constructors. Also in that paper the semantic objects corresponding to the agents contain the description of their transformations under arbitrary name permutations. In that case, however, permutations are only exploited to give a compositional interpretation to the π -calculus constructors: no name deallocation is performed, so the obtained model is intrinsically infinite-state.

References

1. M. Baldamus. Compositional constructor interpretation over coalgebraic models for the π -calculus. In *Proc. CMCS'2000*, ENTCS 33. Elsevier Science, 2000.
2. A. Corradini, M. Große-Rhode, and R. Heckel. Structured transition systems as lax coalgebras. In *Proc. CMCS'98*, ENTCS 11. Elsevier Science, 1998.
3. A. Corradini, R. Heckel, and U. Montanari. From SOS specifications to structured coalgebras: How to make bisimulation a congruence. In *Proc. CMCS'99*, ENTCS 19. Elsevier Science, 1999.
4. G. Ferrari, G. Ferro, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. An Automata Based Verification Environment for Mobile Processes. In *Proc. TACAS'97*, LNCS 1217. Springer Verlag, 1997.
5. G. Ferrari, S. Gnesi, U. Montanari, M. Pistore, and G. Ristori. Verifying Mobile Processes in the HAL Environment. In *Proc. CAV'98*, LNCS 1427, Springer Verlag, 1998.
6. M. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π -calculus. In *Proc. LICS'96*, IEEE. Computer Society Press, 1996.
7. F. Honsell, M. Lenisa, U. Montanari, and M. Pistore. Final Semantics for the pi-calculus. In *PROCOMET'98*. Chapman & Hall, 1998.
8. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (parts I and II). *Information and Computation*, 100(1):1–77, 1992.
9. U. Montanari and M. Pistore. History Dependent Automata. Technical Report TR-11-98. Università di Pisa, Dipartimento di Informatica, 1998.
10. U. Montanari and M. Pistore. Structured Coalgebras and Minimal HD-Automata for the π -Calculus. Technical Report #0006-02. Istituto per la Ricerca Scientifica e Tecnologica, Istituto Trentino di Cultura, 2000. Available at the URL: <http://sra.itc.it/paper.epl?id=MP00>.
11. M. Pistore. *History Dependent Automata*. PhD. Thesis TD-5/99. Università di Pisa, Dipartimento di Informatica, 1999. Available at the URL: http://www.di.unipi.it/phd/tesi/tesi_1999/TD-5-99.ps.gz.
12. J.J.M.M. Rutten. Universal coalgebra: a theory of systems. Technical Report CS-R9652, CWI, 1996. To appear in *Theoretical Computer Science*.
13. I. Stark. A fully abstract domain model for the pi-calculus. In *Proc. LICS'96*, IEEE. Computer Society Press, 1996.

Informative Labeling Schemes for Graphs (Extended Abstract)

David Peleg *

The Weizmann Institute of Science
Department of Computer Science and Applied Mathematics
Rehovot, 76100 Israel
`peleg@wisdom.weizmann.ac.il`

Abstract. This paper introduces and studies the notion of *informative labeling schemes* for arbitrary graphs. Let $f(W)$ be a function on subsets of vertices W . An f *labeling scheme* labels the vertices of a weighted graph G in such a way that $f(W)$ can be inferred efficiently for any vertex subset W of G by merely inspecting the labels of the vertices of W , without having to use any additional information sources.

The paper develops f labeling schemes for some functions f over the class of n -vertex trees, including **SepLevel**, the *separation level* of any two vertices in the tree, **LCA**, the *least common ancestor* of any two vertices, and **Center**, the center of any three given vertices in the tree. These schemes use $O(\log^2 n)$ -bit labels, which is asymptotically optimal.

We then turn to weighted graphs and consider the function **Steiner**(W), denoting the weight of the Steiner tree spanning the vertices of W in the graph. For n -vertex weighted trees with M -bit edge weights, it is shown that there exists a **Steiner** labeling scheme using $O((M + \log n) \log n)$ bit labels, which is asymptotically optimal. In the full paper it is shown that for the class of arbitrary n -vertex graphs with M -bit edge weights, there exists an *approximate-Steiner* labeling scheme, providing an estimate (up to a logarithmic factor) for **Steiner**(W) using $O((M + \log n) \log^2 n)$ bit labels.

1 Introduction

1.1 Problem and Motivation

Network representations have played an extensive role in many domains of computer science, ranging from data structures and graph algorithms to distributed computing and communication networks. The typical goal is to develop various methods and structures for cheaply storing useful information about the network and making it readily and conveniently accessible. This is particularly significant when the network is large and geographically dispersed, and information about its structure must be accessed from various local points in it.

The current paper is dedicated to a somewhat neglected component of network representations, namely, the *labels* assigned to the vertices of the network.

* Supported in part by a grant from the Israel Ministry of Science and Art.

The issue of precisely how are vertex identifiers to be selected is often viewed as minor or inconsequential. For instance, most traditional *centralized* approaches to the problem of network representation are based on storing adjacency information using some kind of a data structure, e.g., an adjacency matrix. Such representation enables one to decide, given the indices of two vertices, whether or not they are adjacent in the network, simply by looking at the appropriate entry in the table. However, note that (a) this decision cannot be made in the absence of the table, and (b) the indices themselves contain no useful information, and they serve only as “place holders,” or pointers to entries in the table, which forms a *global* representation of the network.

In contrast, the notion of *adjacency labeling schemes*, introduced in [2,1], involves using more *informative* and *localized* labeling schemes for networks. The idea is to associate with each vertex a label selected in a such way, that will allow us to infer the adjacency of two vertices *directly* from their labels, without using *any* additional information sources. Hence in essence, this rather extreme approach to the network representation problem *discards* all other components, and bases the entire representation on the set of labels *alone*.

Obviously, labels of unrestricted size can be used to encode any desired information. Specifically, it is possible to encode the entire row i in the adjacency matrix of the graph in the label chosen for vertex i . It is clear, however, that for such a labeling scheme to be useful, it should strive to use relatively *short* labels (say, of length polylogarithmic in n), and yet allow us to deduce adjacencies efficiently (say, within polylogarithmic time). The feasibility of such *efficient* adjacency labeling schemes was explored over a decade ago in [6].

Interest in this natural idea was recently revived by the observation that it may be possible to devise similar schemes for capturing *distance* information. This has led to the notion of *distance labeling schemes*, which are schemes possessing the ability to determine the distance between two vertices efficiently (say, in polylogarithmic time again) given their labels. This notion was introduced in [7], and studied further in [3,4].

The current paper is motivated by the naturally ensuing observation that the ability to decide adjacency and distance are but *two* of a *number* of basic properties a representation may be required to possess, and that many other interesting properties may possibly be representable via an appropriate labeling scheme. In its broadest sense, this observation leads to the general question of developing label-based network representations that will allow retrieving useful information about *arbitrary* functions or substructures in a graph in a *localized* manner, i.e., using only the local pieces of information available to, or associated with, the vertices under inspection, and not having to search for additional *global* information. We term such representations *informative labeling schemes*.

To illustrate this idea, let us consider the class of *rooted trees*. In addition to adjacency or distance, one may be interested in many other pieces of information. For example, it turns out that it is rather easy to encode the ancestry (or descendence) relation in a tree using interval-based schemes (cf. [9]). Another example for a useful piece of non-numeric information is the *least common*

ancestor of two nodes. Moreover, the types of localized information to be encoded by an informative labeling scheme are not limited to *binary* relations. An example for information involving *three* vertices v_1 , v_2 and v_3 is finding their *center*, namely, the unique vertex z connected to them by edge-disjoint paths. More generally, for any subset of vertices W in the tree, one may be interested in inferring $\text{Steiner}(W)$, the weight of their *Steiner tree* (namely, the lightest tree spanning them), based on their labels. The current paper demonstrates the feasibility of informative labeling schemes by providing such schemes for all of the above types of information over the class of rooted trees.

A natural question to ask at this point is whether efficient *exact* informative labeling schemes can be developed for *any* graph family (including, in particular, the family of *all* graphs). Unfortunately, the answer is negative. In [6] it is pointed out that for a family of $\Omega(\exp(n^{1+\epsilon}))$ non-isomorphic n -vertex graphs, for $\epsilon > 0$, any adjacency labeling scheme must use labels whose total combined length is $\Omega(n^{1+\epsilon})$, hence at least one label must be of $\Omega(n^\epsilon)$ bits. In particular, any adjacency labeling scheme for the class of all n -vertex graphs requires labels of size $\Omega(n)$. The same observation carries over to other types of labeling schemes. Hence labeling schemes capturing *approximate* information may be of considerable interest.

The relevance of distance labeling schemes in the context of communication networks has been pointed out in [7], and illustrated by presenting an application of such labeling schemes to distributed connection setup procedures in circuit-switched networks. Some other problems where distance labeling schemes may be useful include memory-free routing schemes, bounded (“time-to-live”) broadcast protocols, topology update mechanisms etc. It is plausible that other types of informative labeling schemes may also prove useful for other applications. In particular, Steiner labeling schemes may be utilized as a basic tool for optimizing multicast schedules and within mechanisms for the selection of subtrees for group communication via communication subtrees, and potentially even for certain information representation problems on the web.

1.2 Related Work

Adjacency labeling systems of general graphs based on Hamming distances were studied in [2,1]. Specifically, in [1] it is shown that it is possible to label the vertices of every n -vertex graph with $2n\Delta$ -bit labels, such that two vertices are adjacent iff their labels are at Hamming distance $4\Delta - 4$ or less of each other, where Δ is the maximum vertex degree in the graph.

An elegant labeling scheme is proposed in [6] for the class of trees using $2 \log n$ -bit labels. It is also shown in [6] how to extend that scheme, and construct $O(\log n)$ adjacency labeling schemes for a number of other graph families, such as bounded arboricity graphs (including, in particular, graphs of bounded degree or bounded genus, e.g., planar graphs), various intersection-based graphs (including interval graphs), and c -decomposable graphs.

It is clear that distance labeling schemes with short labels are easily derivable for highly regular graph classes, such as rings, meshes, tori, hypercubes, and

the like. Whether more general graph classes can be labeled in this fashion is not as clear. It is shown in [7] that the family of n -vertex weighted trees with M -bit edge weights enjoys an $O(M \log n + \log^2 n)$ distance labeling scheme. This scheme is complemented by a matching lower bound given in [3], showing that $\Omega(M \log n + \log^2 n)$ bit labels are necessary for this class. The approach of [7] extends to handle also the class of c -decomposable graphs for constant c , which includes the classes of series-parallel graphs and k -outerplanar graphs, with $c = 2k$. Also, an approximate distance labeling scheme is given in [7] for the class of general weighted graphs.

In [3] it is shown also that n -vertex graphs with a k -separator support a distance labeling with labels of size $O(k \log n + \log^2 n)$. This implies, in particular, that the family of n -vertex planar graphs enjoys such a labeling scheme with $O(\sqrt{n} \log n)$ -bit labels, and the family of n -vertex graphs with bounded tree-width has a distance labeling scheme with labels of size $O(\log^2 n)$. For n -vertex planar graphs, there exists also a lower bound of $\Omega(n^{1/3})$ on the label size required for distance labeling, leaving an intriguing (polynomial) gap. More recently, $O(\log^2 n)$ distance labeling schemes for n -vertex interval and permutation graphs were presented in [4].

1.3 Framework

Let us now formalize the notion of informative labeling schemes.

Definition 1. A vertex-labeling of the graph G is a function L assigning a label $L(u)$ to each vertex u of G . A labeling scheme is composed of two major components. The first is a marker algorithm $\mathcal{M}$, which given a graph G , selects a label assignment $L = \mathcal{M}(G)$ for G . The second component is a decoder algorithm $\mathcal{D}$, which given a set of labels $\hat{L} = \{L_1, \dots, L_k\}$, returns a value $\mathcal{D}(\hat{L})$. The time complexity of the decoder is required to be polynomial in its input size.

Definition 2. Let f be a function defined on sets of vertices in a graph. Given a family $\mathcal{G}$ of weighted graphs, an f labeling scheme for $\mathcal{G}$ is a marker-decoder pair $\langle \mathcal{M}_f, \mathcal{D}_f \rangle$ with the following property. Consider any graph $G \in \mathcal{G}$, and let $L = \mathcal{M}_f(G)$ be the vertex labeling assigned by the marker $\mathcal{M}_f$ to G . Then for any set of vertices $W = \{v_1, \dots, v_k\}$ in G , the value returned by the decoder $\mathcal{D}_f$ on the set of labels $\hat{L}(W) = \{L(v) \mid v \in W\}$ satisfies $\mathcal{D}_f(\hat{L}(W)) = f(W)$.

It is important to note that the decoder $\mathcal{D}_f$, responsible of the f -computation, is independent of G or of the number of vertices in it. Thus $\mathcal{D}_f$ can be viewed as a method for computing f -values in a “distributed” fashion, given any set of labels and knowing that the graph belongs to some specific family $\mathcal{G}$. In particular, it must be possible to define $\mathcal{D}_f$ as a constant size algorithm. In contrast, the labels contain some information that can be pre-computed by considering the whole graph structure.

Clearly, an f -decoder always exists for any graph family if arbitrarily large labels are allowed. Our focus here is on the existence of f labeling schemes which assign labelings with short labels.

For a labeling L for the graph $G = (V, E)$, let $|L(u)|$ denote the number of bits in the (binary) string $L(u)$.

Definition 3. Given a graph G and a marker algorithm $\mathcal{M}$ which assigns the labeling L to G , denote $\mathcal{L}_{\mathcal{M}}(G) = \max_{u \in V} |L(u)|$. For a finite graph family $\mathcal{G}$, set $\mathcal{L}_{\mathcal{M}}(\mathcal{G}) = \max\{\mathcal{L}_{\mathcal{M}}(G) \mid G \in \mathcal{G}\}$. Given a function f and a graph family $\mathcal{G}$, let $\mathcal{L}(f, \mathcal{G}) = \min\{\mathcal{L}_{\mathcal{M}}(\mathcal{G}) \mid \exists \mathcal{D}, \langle \mathcal{M}, \mathcal{D} \rangle \text{ is an } f \text{ labeling scheme for } \mathcal{G}\}$.

1.4 Our Results

This paper starts by introducing and studying f -labeling schemes for three basic functions on the class $\mathcal{T}$ of unweighted trees. For a graph family $\mathcal{G}$, let $\mathcal{G}_n$ denote the subfamily containing the n -vertex graphs of $\mathcal{G}$.

First, we consider the *separation level* function **SepLevel**. The separation level of two vertices in a rooted tree is defined as the depth of their least common ancestor (i.e., its distance from the root of the tree). We show that this function is equivalent to the distance function on the class $\mathcal{T}$ of unweighted trees in terms of its labelability on trees, i.e., it requires labels of size $\Theta(\log^2 n)$, or formally, $\mathcal{L}(\text{SepLevel}, \mathcal{T}_n) = \Theta(\log^2 n)$.

Next, we consider an **LCA** labeling scheme for trees, where $z = \text{LCA}(v, w)$ is the *least common ancestor* of any two vertices v, w . Formally, we assume that each vertex u has a unique *identifier*, denoted $I(u)$, typically of size $O(\log n)$, and the function **LCA** maps the vertex pair (v, w) to the identifier $I(z)$. It is shown that for the class of n -vertex trees, there exists such a labeling scheme using $O(\log^2 n)$ bit labels, and this is asymptotically optimal, i.e., $\mathcal{L}(\text{LCA}, \mathcal{T}_n) = \Theta(\log^2 n)$.

Next, we turn to vertex triples, and consider the **Center** function. The center of three vertices v_1, v_2, v_3 in a tree T is the unique vertex z such that the three paths connecting z to v_1, v_2 and v_3 are edge-disjoint. Here, too, we show the existence of an (asymptotically optimal) **Center** labeling scheme using $O(\log^2 n)$ bit labels, i.e., $\mathcal{L}(\text{Center}, \mathcal{T}_n) = \Theta(\log^2 n)$ as well.

We then turn to *weighted* graphs. For a graph family $\mathcal{G}$, let $\mathcal{G}_{n,M}$ denote the subfamily containing the n -vertex graphs of $\mathcal{G}$ with M -bit edge weights. We consider **Steiner** labeling schemes for graphs. Given a subset W of vertices in G , a *Steiner tree* $T_S(W)$ for W is a minimum weight tree spanning all the vertices of W (and perhaps some other vertices as well) in G . The *Steiner weight* of W , denoted **Steiner**(W), is the weight of the Steiner tree $T_S(W)$. Using the **LCA** labeling scheme, we show that the **Steiner** weight function has an $O((M + \log n) \log n)$ size labeling scheme on the class $\mathcal{T}_{n,M}$ of weighted n -vertex trees with M -bit edge weights, and this is asymptotically optimal, i.e., $\mathcal{L}(\text{Steiner}, \mathcal{T}_{n,M}) = \Theta((M + \log n) \log n)$.

Finally, in the full paper (see [8]) we consider the class of *arbitrary* weighted graphs $\mathcal{G}$. We show that an exact **Steiner** labeling scheme for the class of arbitrary weighted graphs $\mathcal{G}_{n,M}$ clearly requires at least $\Omega(M + n)$ -bit labels. We therefore turn to labeling schemes providing approximate information, and show that for the class of arbitrary n -vertex graphs with M -bit edge weights, there exists an approximate **Steiner** labeling scheme (up to logarithmic factor) using $O((M + \log n) \log^2 n)$ bit labels.

2 SepLevel Labeling Schemes

We start with a **SepLevel** labeling scheme for trees. Consider a rooted tree T with root r_0 . The depth of a vertex $v \in T$, denoted $\text{depth}(v)$, is its distance $\text{dist}(v, r_0)$ from the root r_0 . Two vertices $v, w \in T$ are said to have *separation level* $\text{SepLevel}(v, w) = \ell$ if their least common ancestor z has depth $\text{depth}(z) = \ell$. We now claim¹ that for the class $\mathcal{T}$ of unweighted trees, distance labeling and **SepLevel** labeling require the same label size up to an additive logarithmic² term.

Lemma 1. $\mathcal{L}(\text{dist}, \mathcal{T}_n) - \log n \leq \mathcal{L}(\text{SepLevel}, \mathcal{T}_n) \leq \mathcal{L}(\text{dist}, \mathcal{T}_n) + \log n$.

Based on the upper and lower bounds of [7,3] for distance labeling schemes for trees, we get

Corollary 1. *For the class of n -vertex trees $\mathcal{T}_n$, $\mathcal{L}(\text{SepLevel}, \mathcal{T}_n) = \Theta(\log^2 n)$.*

3 LCA Labeling Schemes

We now turn to developing an **LCA** labeling scheme for trees, where $z = \text{LCA}(v, w)$ is the *least common ancestor* of any two vertices v, w . As mentioned earlier, this requires us to assume that each vertex u has a unique *identifier*, denoted $I(u)$, of size $O(\log n)$, and the function **LCA** maps the vertex pair (v, w) to the identifier $I(z)$.

For every vertex v in the tree, let $T(v)$ denote the subtree of T rooted at v . For $0 \leq i \leq \text{depth}(v)$, denote v 's ancestor at level i of the tree by $\gamma_i(v)$. In particular, $\gamma_0(v) = r_0$ and $\gamma_{\text{depth}(v)}(v) = v$.

Definition 4. *A nonroot vertex v with parent w is called small if its subtree, $T(v)$, contains at most half the number of vertices contained in its parents' subtree, $T(w)$. Otherwise, v is large.*

For every vertex v , the “small ancestor” levels of v are the levels above it in which its ancestor is small, $\text{SAL}(v) = \{i \mid 1 \leq i \leq \text{depth}(v), \gamma_i(v) \text{ is small}\}$, and the small ancestors of v are $\text{SA}(v) = \{\gamma_i(v) \mid i \in \text{SAL}(v)\}$.

The labels are constructed as follows. As a preprocessing step, assign each v an *interval* $\text{Int}(v)$ as in the interval labeling scheme of [9], in addition to its identifier $I(v)$. This scheme is based on the following two steps. First, construct a *depth-first* numbering of the tree T , starting at the root, and assign each vertex $u \in T$ a depth-first number $\text{DFS}(u)$. Then, label a vertex u by the interval $\text{Int}(u) = [\text{DFS}(u), \text{DFS}(w)]$, where w is the last descendent of u visited by the DFS tour. The resulting interval labels are of size $O(\log n)$. What makes these interval labels useful for our purposes is the fact that they enjoy the following

¹ Proofs are omitted, see [8].

² For clarity of presentation we ignore rounding issues in stating our claims. For instance, here and in several other places, $\log n$ stands for $\lceil \log n \rceil$.

important property: *For every two vertices u and v of the tree T , $\text{Int}(v) \subseteq \text{Int}(u)$ iff v is a descendent of u in T .*

For a vertex v and $1 \leq i < \text{depth}(v)$, the i -triple of v is

$$Q_i(v) = \left\langle \langle i-1, I(\gamma_{i-1}(v)) \rangle, \langle i, I(\gamma_i(v)) \rangle, \langle i+1, I(\gamma_{i+1}(v)) \rangle \right\rangle.$$

In the second and main stage, we assign for each vertex v the label

$$L(v) = \langle I(v), \text{Int}(v), \{Q_i(v) \mid 1 \leq i < \text{depth}(v), i \in \text{SAL}(v)\} \rangle.$$

Let us now describe the LCA-decoder $\mathcal{D}_{\text{LCA}}$ which, given two vertex labels $L(v)$ and $L(w)$, infers the identifier $I(z)$ of their least common ancestor $z = \text{LCA}(v, w)$.

Decoder $\mathcal{D}_{\text{LCA}}$

1. If $\text{Int}(w) \subseteq \text{Int}(v)$ then return $I(v)$. /* v is an ancestor of w */
2. If $\text{Int}(v) \subseteq \text{Int}(w)$ then return $I(w)$. /* w is an ancestor of v */
3. Extract from $L(v)$ and $L(w)$ the sets $\text{SAL}(v)$, $\text{SAL}(w)$, $\text{SA}(v)$ and $\text{SA}(w)$.
4. Let α be the highest level vertex in $\text{SA}(v) \cap \text{SA}(w)$. Let K be its level, i.e., $\alpha = \gamma_K(v) = \gamma_K(w)$. /* α = least common small ancestor of v and w */
5. If $\gamma_{K+1}(v) \neq \gamma_{K+1}(w)$ then return $I(\alpha)$.
6. /* $\gamma_{K+1}(v) = \gamma_{K+1}(w)$ is also a common (yet large) ancestor of v and w */
Let $i_v = \min\{i \in \text{SAL}(v) \mid i > K\}$, $i_w = \min\{i \in \text{SAL}(w) \mid i > K\}$,
 $i_m = \min\{i_v, i_w\}$.
7. Extract $\gamma_{i_m-1}(v)$ from the i_m -triple $Q_{i_m}(v)$; Return $I(\gamma_{i_m-1}(v))$.

Let us now prove the correctness of the labeling scheme. It is immediate to observe that if v is an ancestor of w or vice versa, then Step 1 of the decoder $\mathcal{D}_{\text{LCA}}$ correctly finds $\text{LCA}(v, w)$. Hence hereafter we assume that neither of the above holds, i.e., $\text{LCA}(v, w)$ is neither v nor w .

For the remainder of this section, denote $z = \text{LCA}(v, w)$, and let its level be $t = \text{depth}(z)$. Let x be the child of z on the path to v , and let y be the child of z on the path to w .

Lemma 2. $\text{SA}(v) \cap \text{SA}(w) = \text{SA}(z)$.

Let K and $\alpha = \gamma_K(v) = \gamma_K(w)$ be the level number and vertex selected in Step 4 of the algorithm. By the previous lemma, $\alpha \in \text{SA}(z)$, so $K \leq t$ and α is small, and hence $K \in \text{SAL}(z)$.

Now observe that if $K = t$ then we are done, since in this case the test done in Step 5 will necessarily succeed, and subsequently the algorithm will return $I(\alpha)$, which is the correct answer. Hence it remains to handle the case when $K < t$. In this case, the test of Step 5 will fail, and the execution will reach Steps 6 and 7. It is obvious from the definitions that each vertex has at most one large child. Consequently, as $x \neq y$ and both are the children of the same parent z , at least one of them is small, hence $x \in \text{SA}(v)$ or $y \in \text{SA}(w)$.

Lemma 3. $i_v, i_w \geq t + 1$.

Lemma 4. (1) If $x \in \text{SA}(v)$ then $i_v = t + 1$, and (2) if $y \in \text{SA}(w)$ then $i_w = t + 1$.

Combining the last three lemmas yields $i_m = t + 1$. Hence the output returned by the algorithm in Step 7 is the correct one, $z = \gamma_t(v)$.

Lemma 5. *For every two vertices v and w , the decoder $\mathcal{D}_{\text{LCA}}$ correctly deduces $\text{LCA}(v, w)$ given $L(v)$ and $L(w)$.*

It is obvious from the definitions that in an n -vertex tree, every vertex v has at most $\log n$ small ancestors, i.e., $|SA(v)| \leq \log n$. It follows that each vertex v has at most $\log n$ i -triples $Q_i(v)$. The size of the resulting labels thus depends on the size of the identifiers used by the scheme. In particular, let $g(n)$ denote the maximum size of an identifier assigned to any vertex in any n -vertex tree. Clearly $g(n) = \Omega(\log n)$, hence each triple requires $O(g(n))$ bits, and the entire label is of size $O(g(n) \log n)$.

Theorem 1. *$\langle \mathcal{D}_{\text{LCA}}, \mathcal{M}_{\text{LCA}} \rangle$ is an LCA labeling scheme with labels of size $O(g(n) \log n)$ for the class $\mathcal{T}_n$ of n -vertex trees with identifiers of size $g(n)$.* ■

Since $\log n$ -bit identifiers can always be chosen, we get that $\mathcal{L}(\text{LCA}, \mathcal{T}_n) = O(\log^2 n)$. Note that this is optimal, in the following sense.

Lemma 6. *If $\mathcal{T}_n$ has an LCA labeling scheme with $l(n) \cdot g(n)$ -bit labels over $g(n)$ -bit identifiers, then it has a SepLevel labeling scheme with $l(n) \cdot (g(n) + \log n)$ -bit labels.*

Since $g(n) = \Omega(\log n)$, Corollary 1 implies

Corollary 2. $\mathcal{L}(\text{LCA}, \mathcal{T}_n) = \Theta(\log^2 n)$.

4 Center Labeling Schemes

For every three vertices v_1, v_2, v_3 in a tree T , let $\text{Center}(v_1, v_2, v_3)$ denote their *center*, namely, the unique vertex z such that the three paths connecting z to v_1 , v_2 and v_3 are edge-disjoint (in fact, also vertex-disjoint except at z). We now show that an LCA-marker can serve also as a Center-marker, provided that the identifiers it uses are themselves ancestry and depth labelings, namely, the identifier $I(v)$ contains v 's level $\text{depth}(v)$ and any two identifiers $I(v)$ and $I(w)$ allows us to deduce whether one of the two vertices is an ancestor of the other. As mentioned earlier, both requirements are achievable using identifiers of size $O(\log n)$. Hence the Center-marker $\mathcal{M}_{\text{Center}}$ will first pick such identifiers for the vertices, and then invoke the LCA-marker $\mathcal{M}_{\text{LCA}}$ described in the previous section for generating the labels.

We next present an algorithm for computing $\text{Center}(v_1, v_2, v_3)$ given the labels $L(v_1)$, $L(v_2)$ and $L(v_3)$. For $1 \leq i < j \leq 3$, denote $z_{i,j} = \text{LCA}(v_i, v_j)$.

Decoder $\mathcal{D}_{\text{Center}}$

1. Compute $I(z_{1,2})$, $I(z_{1,3})$ and $I(z_{2,3})$.
2. If the three LCA's coincide then return $I(z_{1,2})$.
3. If exactly two LCA's coincide, say, $z_{1,3} = z_{1,2}$, then return the third, $I(z_{2,3})$.

We rely on the easy to verify facts that for every three vertices v_1, v_2, v_3 in a rooted tree, at least two of the three LCA's $z_{1,2}$, $z_{1,3}$ and $z_{2,3}$ must coincide, and that if $z_{1,3} = z_{1,2} \neq z_{2,3}$, then $z_{2,3}$ is a descendent of $z_{1,3}$. As an easy corollary we get that for every three vertices v_1, v_2 and v_3 , the **Center**-decoder $\mathcal{D}_{\text{Center}}$ correctly deduces $\text{Center}(v_1, v_2, v_3)$ given $L(v_i)$ for $i = 1, 2, 3$.

Theorem 2. $\mathcal{L}(\text{Center}, \mathcal{T}_n) = O(\log^2 n)$. ■

Finally, let us record the following fact for later use.

Lemma 7. *The **Center** labeling scheme allows us also to deduce the distance between $z = \text{Center}(v_1, v_2, v_3)$ and each v_i , $1 \leq i \leq 3$.*

5 Steiner Labeling Schemes

Our final section concerns weighted graphs. For a vertex set W in a weighted graph G , the *Steiner tree* $T_S(W)$ is the minimum-weight subtree of G spanning the vertices of W , and its weight is denoted $\text{Steiner}(W)$. A **Steiner** labeling scheme deduces $\text{Steiner}(W)$ given the labels $\{L(v) \mid v \in W\}$.

We now show that the **Center**-marker $\mathcal{M}_{\text{Center}}$ presented in the previous section can serve also as a **Steiner**-marker within a **Steiner** labeling scheme. In particular, we rely also on the fact that in the labelings produced by the **Center**-marker $\mathcal{M}_{\text{Center}}$, the identifiers $I(v)$ of every vertex v provide $\text{depth}(v)$.

Dealing with weighted graphs requires us, in particular, to use weighted measures of distance and depth. This means that when employing the **Center** labeling scheme of the previous section, which in turn makes use of our other schemes, the distance and depth functions used by the schemes must be the weighted ones. While this does not require any other change in the schemes, it does have some immediate implications on the size of the resulting labels, as explained later on.

Given a **Center**-marker $\mathcal{M}_{\text{Center}}$ as in the previous section, and taking the **Steiner**-marker to be $\mathcal{M}_{\text{Steiner}} = \mathcal{M}_{\text{Center}}$, we now present a **Steiner**-decoder $\mathcal{D}_{\text{Steiner}}$ for computing the weight $\text{Steiner}(W)$ of the Steiner tree $T_S(W)$ for any vertex set W in T , given as input the labels $L(v)$ for every $v \in W$.

Let us first consider the case when $|W| = 3$, or $W = \{v_1, v_2, v_3\}$. In this case, the **Steiner**-decoder $\mathcal{D}_{\text{Steiner}}$ simply deduces the center $z = \text{Center}(v_1, v_2, v_3)$, calculates the distances $d_i = \text{dist}(v_i, z)$ for $1 \leq i \leq 3$ as in Lemma 7, and returns $\omega(W) = d_1 + d_2 + d_3$. Now suppose that W contains more than three vertices, $W = \{v_1, \dots, v_q\}$ for $q > 3$. For every $3 \leq k \leq q$, let $W_k = \{v_1, \dots, v_k\}$. Given the set W , the **Steiner**-decoder $\mathcal{D}_{\text{Steiner}}$ works iteratively, starting by computing $\omega(W_3)$ and adding the remaining vertices one at a time, computing $\omega(W_k)$ for $k = 4, \dots, q$.

Decoder $\mathcal{D}_{\text{Steiner}}$

1. Deduce the center $z = \text{Center}(v_1, v_2, v_3)$.
2. Calculate $d_i = \text{dist}(v_i, z)$ for $1 \leq i \leq 3$ (as in Lemma 7), and let $\omega(W_3) = d_1 + d_2 + d_3$.

3. **For** $k = 4$ to q **do**:

- a) For every $1 \leq i < j \leq k$, compute $z_{i,j} = \text{Center}(v_i, v_j, v_{k+1})$.
- b) For every $1 \leq i < j \leq k$, compute $d_{i,j} = \text{dist}(z_{i,j}, v_{k+1})$ (again, as in Lemma 7).
- c) Let $1 \leq i' < j' \leq k$ be the pair minimizing $d_{i,j}$.
- d) Let $\omega(W_{k+1}) = \omega(W_k) + d_{i',j'}$.

4. Return $\omega(W_q)$.

Definition 5. For a subtree T' and a vertex v in T , let $p(v, T')$ denote the (unique) shortest path connecting v to some vertex of T' .

Lemma 8. For every set of vertices $W = \{v_1, \dots, v_k\}$ and vertex $v \notin W$, there exists a pair of vertices $v_i, v_j \in W$, connected by a path $P_{i,j}$ in T , such that $p(v, T_S(W)) = p(v, P_{i,j})$.

It follows that for every set of vertices W in T , the **Steiner**-decoder $\mathcal{D}_{\text{Steiner}}$ correctly deduces $\omega(W)$ given $L(v)$ for every $v \in W$. As mentioned earlier, label sizes may be somewhat larger in the weighted case. Specifically, if M -bit edge weights are used, then the $\text{depth}(v)$ field in the identifier $I(v)$ may require $\Theta(M + \log n)$ bits in the worst case. On the other hand, as $\text{dist}(v_1, v_2, T) = \text{Steiner}(W)$ for any pair of vertices $W = \{v_1, v_2\}$, the lower bound of $\mathcal{L}(\text{dist}, \mathcal{T}_{n,M}) = \Theta(M \log n + \log^2 n)$ established in [3] extends to the **Steiner** function as well. This yields the following result.

Theorem 3. $\mathcal{L}(\text{Steiner}, \mathcal{T}_{n,M}) = \Theta(M \log n + \log^2 n)$. ■

Acknowledgements. I am grateful to Michal Katz and Nir Katz for their helpful comments and suggestions.

References

1. M.A. Breuer and J. Folkman. An unexpected result on coding the vertices of a graph. *J. Mathemat. Analysis and Applic.*, 20:583–600, 1967.
2. M.A. Breuer. Coding the vertexes of a graph. *IEEE Trans. on Information Theory*, IT-12:148–153, 1966.
3. C. Gavoille, D. Peleg, S. Pérennes and R. Raz. Distance labeling in graphs. Unpublished manuscript, September 1999.
4. M. Katz, N.A. Katz and D. Peleg. Distance labeling schemes for well-separated graph classes. In *Proc. 17th STACS*, pages 516–528, 2000.
5. L. Kou, G. Markowsky and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.
6. S. Kannan, M. Naor and S. Rudich. Implicit representation of graphs. In *Proc. 20th STOC*, pages 334–343, May 1988.
7. D. Peleg. Proximity-preserving labeling schemes and their applications. In *Proc. 25th WG*, pages 30–41, June 1999.
8. D. Peleg. Informative labeling schemes for graphs. Technical Report MCS00-05, The Weizmann Institute of Science, 2000.
9. N. Santoro and R. Khatib. Labelling and implicit routing in networks. *The Computer Journal*, 28:5–8, 1985.

Separation Results for Rebound Automata

Holger Petersen

Institut für Informatik, Universität Stuttgart
Breitwiesenstr. 20–22, D-70565 Stuttgart**
petersen@informatik.uni-stuttgart.de

Abstract. We show that the class of languages accepted by nondeterministic two-dimensional rebound automata properly contains the class of languages accepted by deterministic rebound automata. Further we separate the class of languages accepted by deterministic one-way one counter automata from the languages accepted by rebound automata, strengthening previous results. The language separating these classes is in fact deterministic linear, which improves the known result that there is a context-free language not accepted by rebound automata.

1 Introduction

A classical result in the theory of two-dimensional automata says that non-determinism is more powerful than determinism for finite state devices. This separation, which marks an important difference to one-dimensional automata, is due to Blum and Hewitt [1] (see also [10, Theorem 4.3.4]).

A major problem in the investigation of sets of two- or higher-dimensional input objects is however that the classes arising from devices with varying capabilities cannot directly be compared with their one-dimensional counterparts. This motivated the restriction to *rebound automata*, two-dimensional finite automata operating on a square grid that contains a one-dimensional input string in the first row and blank symbols otherwise.

Deterministic rebound automata were introduced and investigated by Sugata, Umeo, and Morita. An extension to nondeterministic automata was already briefly mentioned in their early paper [12]. The question whether the trivial inclusion between the classes of languages accepted by deterministic and nondeterministic rebound automata is proper, appeared explicitly in [6,13,9]. Note that the separation result for general automata mentioned above does not apply to rebound automata because the set of inputs separating these classes makes essential use of non-blank symbols on other sections of the tape than the first row.

The main result of the present paper is a separation of the classes of languages accepted by deterministic and nondeterministic rebound automata, solving the long standing open problem. Note that by including the blank portions of the

** Current address: The Academic College of Tel-Aviv-Yaffo, 4 Antokolsky St., Tel-Aviv 64044, Israel.

input we obtain a set of two-dimensional patterns that separates deterministic and nondeterministic finite automata. This strengthens the classical result of Blum and Hewitt [1].

Further we define a language that can be accepted by a one-way counter automaton, but not by an even nondeterministic rebound automaton, solving another open problem from [6]. Since this language is a member of the intersection between deterministic one-way counter languages and deterministic linear context-free languages, this result at the same time strengthens known separations from [11,9] and [6], where nondeterministic rebound automata were separated from two-way counter automata and context-free languages.

2 The General Strategy

The technique that applies to many variants of two-dimensional automata is an analysis of the flow of information across the borders between two portions or groups of portions of the input. In our proofs we will assume that there is a certain enumeration of the borders between tape cells in a left to right and bottom to top fashion. It is then possible to specify a subset of these borders, which we will call the *interface* of the portions. Provided that the input is sufficiently long this interface will be present regardless of the exact length of the input.

In the separation of two computational models A and B the important ingredients are the choice of a suitable witness language L and interfaces such that the following conditions are satisfied:

1. Model A can transfer a sufficient amount of information across the interfaces between the portions for deciding L .
2. Model B is not able to provide this information.

After fixing the separating language L and describing an algorithm for accepting L with a device of type A we will therefore have to specify an interface (depending on the size of an automaton of type B) and argue that for some input the automaton cannot make the correct decision. This is done by gluing together at the interface portions of the input that do not constitute a member of L .

As a simplification of our discussion we may without loss of generality require that all rebound automata, if they accept, do so with their head on the upper left cell of the tape.

3 Separating Determinism from Nondeterminism for Rebound Automata

The language V used in the separation of deterministic and nondeterministic rebound automata is defined as follows:

$$V = \{a^j b^k 0^{7 \cdot 2^r} y_{2^r - j - k} \cdots y_1 \mid \\ \forall 1 \leq i \leq 2^r - j - k : y_i \in \{0, 1\}, \quad y_{(2j+1)2^k} = 1\}$$

Lemma 1. *The language V can be accepted by a nondeterministic rebound automaton*

Proof. We will sketch an algorithm carried out by a nondeterministic rebound automaton A accepting V . In the following we will often refer to numbers represented as column positions of the input head on the first row of the input tape. We frequently make use of a technique of multiplying or dividing these numbers by constants. This is done by moving diagonally across the input with a ratio of horizontal and vertical moves that results in the intended factor. The remainders of divisions are stored in the finite control.

First A checks that the top most row of its input contains a 's followed by b 's and then a sequence of 0's and 1's.

It checks that its input length is a power of 2 of the form $m = 2^{r+3}$. It does so by successively dividing column positions by 2, checking that this eventually results in 1, and verifying that the number of iterations is at least 3.

Next A moves its head to the leftmost 1 (if any), subtracts $m - \frac{1}{8}m = 7 \cdot 2^r$ from the column position and checks that the scanned symbol is 0.

Then A computes numbers $p_i = (2j + 1)2^i$ as column positions for $i = 0, 1, 2, \dots$ (as long as this is possible) and uses these numbers in order to access y_{p_i} . For some nondeterministically chosen p_i such that $y_{p_i} = 1$ the automaton stops this process (if no such p_i exists, A rejects). Note that $p_i \leq 2^r$. Now A adds $\frac{1}{2}m = 2^{r+2}$ to the column position and repeatedly divides the current position by 2 until a remainder of 1 occurs.

Then A multiplies by 2, increments by 3, and repeatedly multiplies by 2 until a value greater or equal 2^{r+2} is reached. Then A subtracts 2^{r+2} and divides by 2. This process is repeated until the last division results in a non-zero remainder.

Note that the arithmetic operations on the input head positions transform $(2j + 1)2^i$ into $(2(j + 1) + 1)2^{i-1}$ for each $i \geq 1$. For $j = 0$ the first transformation yields $3 \cdot 2^{i-1} < 2^{r+2}$, for $j > 0$ we have $(2(j + 1) + 1)2^{i-1} = ((2j + 1) + 2)2^{i-1} \leq (2j + 1)2^i < 2^{r+2}$. Therefore the counting will be carried out correctly.

The final value computed as A 's head position is $j + i + 1$. Thus A can use this value in order to verify that it guessed $p_k = (2j + 1)2^k$ correctly by checking that the symbol at position $j + i + 1$ is the leftmost 0.

In summary A checks syntactical correctness of the input and that the flag y_{p_k} is 1.

□

Lemma 2. *The language V cannot be accepted by deterministic rebound automata.*

Proof. We will assume that there exists a deterministic rebound automaton R with set of states Q that decides V . Depending on the number of states of R we will choose a set of possible input strings such that R cannot correctly decide every one of them and therefore no R of the required kind can exist.

We focus on input strings with a prefix $w = a^j b^k 0^{n-j-k}$ with $n \geq j + k$. Notice that the number of different prefixes of this kind is

$$\frac{n^2 + 3n + 2}{2} \geq \frac{n^2}{2}.$$

Split

*	*	*	·	*	·	*	*
*	x_1	x_2	·	x_n	·	x_m	*
*			·		·		*
*			·		·		*
·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·
*			·		·		*
*	*	*	·		·	*	*

into

*	*	*	·	*
*	x_1	x_2	·	x_n

·	*	*
·	x_m	*

*			·		·		*
*			·		·		*
·	·	·	·	·	·	·	·
·	·	·	·	·	·	·	·
*			·		·		*
*	*	*	·		·	*	*

where each $x_i \in \{a, b, 0, 1\}$, n is fixed, and m varies. Let $v = x_{n+1} \cdots x_m$ (the entire input is therefore wv).

The interface I consists of the $n + 3$ borders between tape cells that separate the two parts of the input. Define the partial function $C_v : I \times Q \rightarrow I \times Q$ by

$$C_v(i, p) = (i', q)$$

if R will return to position i' of I and enter state q if it leaves w 's part of the input at position i of I and entering state p . Observe that if $C_v = C_{v'}$ then wv and wv' are either both accepted or rejected. This is because the segments on w of an accepting computation for wv can be combined with the computations on the remaining input to form an accepting computation on wv' and vice versa.

The number of different partial functions C_v for an automaton with $s = |Q|$ states is bounded by $(s(n + 3) + 1)^{s(n+3)}$. Since an automaton accepting a non-trivial language needs at least one accepting and one non-accepting state we may assume $s \geq 2$.

Via the encoding as $(2j + 1)2^k$ each v defines a set $M_v \subseteq \{(j, k) \mid j, k \leq n\}$ in the sense that $(j, k) \in M_v$ if and only if $a^j b^k 0^{n-j-k} v \in V$.

In order to show that automaton R cannot exist it is sufficient to find an n such that the number of subsets M_v exceeds the number of functions C_v . Then we will have some v, v' with $C_v = C_{v'}$ for $M_v \neq M_{v'}$ and we can choose $(j, k) \in (M_v \setminus M_{v'}) \cup (M_{v'} \setminus M_v)$. Now R accepts either both $a^j b^k 0^{n-j-k} v$ and $a^j b^k 0^{n-j-k} v'$ or none of them, contrary to the definition of V .

We claim that $n = s^8$ has the desired property, since:

$$\begin{aligned}
 (s(s^8 + 3)) \cdot \log_2(s(s^8 + 3) + 1) &\leq 2s^9 \cdot \log_2(2s^9 + 1) \\
 &\leq 2s^9 \cdot \log_2(4s^9) \\
 &\leq 2s^9 \cdot (2 + 9 \log_2 s) \\
 &\leq 20s^{10} \\
 &< 32s^{10} \\
 &\leq \frac{s^6}{2} s^{10} \\
 &= \frac{(s^8)^2}{2}
 \end{aligned}$$

Therefore $(s(n+3)+1)^{s(n+3)}$ (the upper bound on the number of functions C_v) is smaller than $2^{n^2/2}$ (the lower bound on number of different sets M_v). $\square$

Combining the two lemmas we have solved the open problem from [6,13,9]:

Theorem 1. *The class of languages accepted by deterministic rebound automata is properly included in the class of languages accepted by nondeterministic rebound automata.*

4 Separating One-Way Counter Automata and Rebound Automata

We first give the definition of the witness language $W \subseteq \{0, 1, \$, \#\}^*$ that will separate counter automata and rebound automata. This language is a subset of the regular language $W' = ((0+1)\$0^*\$)^*\#(0+1)^*$ and is defined as:

$$\begin{aligned}
 W &= \{x_1\$0^{m_1}\$ \cdots \$x_n\$0^{m_n}\#\#y_0 \cdots y_r \mid \\
 &\quad \forall i \leq n : x_i \in \{0, 1\}, \\
 &\quad \forall i \leq r : y_i \in \{0, 1\}, \\
 &\quad y_p = 1, \text{ where } p = \sum_{i \in \{1, \dots, n\} : x_i = 1} m_i\}
 \end{aligned}$$

Intuitively words in this language have flags x_i that select numbers m_i which are added to form a number ℓ . The segment after $\#$ encodes a set of numbers $M = \{i \mid y_i = 1\}$ with the property that $\ell \in M$.

Lemma 3. *The language W can be accepted by a deterministic one-way one counter automaton making at most one reversal on its counter.*

Proof. We will describe an algorithm that can be carried out by an automaton A as required by the lemma. We tacitly assume that A checks membership in the regular superset W' of W .

Starting with an empty counter, A adds each m_i to the value currently stored on the counter if and only if the corresponding x_i is 1. Clearly this can be done without reversals. After reading $\#$ the automaton accepts if it reads a 1 and the counter has a value of zero. For every symbol processed it decrements its counter. If the counter would become negative by a decrement operation or if all y_i 's have been processed without reaching zero on the counter, the input is rejected.

Before reading $\#$ the counter is incremented, after reading this symbol it is decremented and therefore A makes at most one reversal. □

Lemma 4. *The language W cannot be accepted by nondeterministic rebound automata.*

Proof. We will assume that there exists a nondeterministic rebound automaton R that accepts W and derive a contradiction.

For each $n \geq 1$ we restrict our attention to a certain class of possible input strings, namely strings of the following form:

$$x_1 \$ 0^{2^0} \$ \dots \$ x_n \$ 0^{2^{n-1}} \$ \# y_0 \dots y_r \tag{1}$$

with $x_1, \dots, x_n \in \{0, 1\}$ and $r = 2^n - 1$.

Split

*	*	*	.	*	*	*	.	.	.	*	*	*	.	*	.	.	.	*
*	x_1	\$	.	\$	x_2	\$	.	.	.	\$	x_n	\$	.	#	.	.	.	*
*			.				.	.	.				.		.	.	.	*
*			.				.	.	.				.		.	.	.	*
*			.				.	.	.				.		.	.	.	*
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
*			.				.	.	.				.		.	.	.	*
*			.				.	.	.				.		.	.	.	*
*			.				.	.	.				.		.	.	.	*
*	*	*	.	*	*	*	.	.	.	*	*	*	.	*	.	.	.	*

with $x_1, \dots, x_n \in \{0, 1\}$ into

*	*			*			*
*	x_1			x_2	.	.	x_n

		*	.	*		*	.	.	.	*		*	.	.	.	*
		\$	.	\$		\$	.	.	.	\$		\$	.	#	.	*
*			.			.	.	.				.		.	.	*
*			.			.	.	.				.		.	.	*
*			.			.	.	.				.		.	.	*
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
*			.			.	.	.				.		.	.	*
*			.			.	.	.				.		.	.	*
*			.			.	.	.				.		.	.	*
*	*	*	.	*	*	*	.	.	.	*	*	*	.	*	.	*

The interface I between the portions including the x 's and the rest of the input has size $5n - 1$. Suppose rebound automaton R has a set of internal states Q with $s = |Q| \geq 2$. For a fixed input the possible computations of R between two visits of portions including some x 's are determined by a relation $C \subseteq (I \times Q) \times (I \times Q)$, where $((i, p), (j, q)) \in C$ if and only if R leaving one of the portions containing $x_1, \dots, x_n$ at position i of the interface and entering state p is able to return to the interface and cross it at position j entering state q .

Note that inputs of the form (1) having the same $x_1, \dots, x_n$, and giving rise to the same relation C will either all be accepted, or none of them will be in the language accepted by R .

There are at most $2^{s^2(5n-1)^2}$ different relations that can occur for automaton R on an input of the form (1). We will now argue that this number is not sufficient for deciding W correctly.

The sequence $x_1, \dots, x_n$ specifies a number in the set $M = \{0, \dots, 2^n - 1\}$. Any subset of M can be encoded by a sequence $y_0 \dots y_r$. There are 2^{2^n} subsets of M . For $n \geq 12s$ this number exceeds the number of different relations. Therefore two subsets $M_1, M_2 \subseteq M$ with $M_1 \neq M_2$ admit the same relation C defined above. Take some number $\ell \in (M_1 \setminus M_2) \cup (M_2 \setminus M_1)$. Let $x_1, \dots, x_n$ specify ℓ and combine these portions with trailing strings that encode M_1 and M_2 , respectively. Now R either accepts both of these inputs or none of them, which contradicts the definition of W . We conclude that there is no rebound automaton R accepting W .

□

Rebound automata accept languages like

$$P = \{w\$w^R \mid w \in \{0, 1\}^*\},$$

where w^R is the reversal of w [12]. By considering the set of possible configurations that a counter automaton can reach when reading the $\$$ it is clear that no

deterministic one-way one counter automaton can accept P . This statement also follows from the more general result that deterministic one-way multi-counter automata need exponential time for accepting P [4] and deterministic one-way one-counter automata accept in linear time.

From the two preceding lemmas and this fact the following theorem can be deduced.

Theorem 2. *The class of languages accepted by nondeterministic rebound automata is incomparable to the class of languages accepted by deterministic one-way counter automata.*

This answers the open problem from [6], whether there is a language accepted by a nondeterministic one-way (or on-line) one-counter machine, but not by any nondeterministic rebound automaton.

Since the language W can be accepted by a deterministic one-turn one-counter automaton, it is a deterministic linear context-free language and we are able to strengthen the result from [6] that some context-free language cannot be accepted by rebound automata (the witness language in [6] is linear, but not deterministic context-free).

Theorem 3. *There is a deterministic linear context-free language not accepted by any nondeterministic rebound automaton.*

Together with the simulation of deterministic rebound automata by deterministic two-way counter automata [7] we also obtain the result from [9].

Corollary 1 ([9]). *The class of languages accepted by deterministic rebound automata is properly included in the class of languages accepted by deterministic two-way one-counter automata.*

Rebound automata have been extended by adding a finite number of markers or pebbles, that the automata can place on the input tape and later may recognize, pick up, and redistribute. It is known that k markers can simulate $2k$ heads of a one-dimensional finite automaton [12]. We observe that the separation of counter automata and rebound automata applies to automata with two heads, since a head can simulate a counter that is bounded by the input length. This leads to an alternative proof of Theorem 2.2 of [6].

Corollary 2 ([6]). *There is a language accepted by a deterministic rebound automaton with a marker, but by no nondeterministic rebound automaton without markers.*

Note that a separation of the deterministic models also follows from [3].

5 Conclusion and Open Problems

We could establish that nondeterminism adds power to finite rebound automata. The witness language for this separation as well as the languages in the second

result and in a previous paper [9] are not very natural. A much more typical example that seems to separate counter automata and rebound automata is

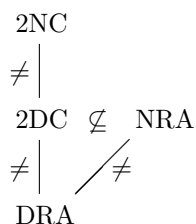
$$L = \{w \mid w \in \{0, 1\}^*, |w|_0 = |w|_1\}$$

(here $|w|_x$ denotes the number occurrences of symbol x in w). A deterministic one-way counter automaton can accept this language in an obvious way. As suggested by Sugata, Umeo, and Morita [12] the language L probably cannot be accepted by rebound automata, but still there is no proof of this conjecture.

In a similar way as two-dimensional finite automata without markers, non-deterministic and deterministic two-dimensional automata with a single marker have been separated, see [5] for the references. Here the separating set consists of tapes over $\{0, 1\}$ such that the upper and the lower halves of the input are identical (in fact the language consists of non-square tapes, but the result might as well be based on square input tapes of even side-length). The ability to place non-blank symbols at arbitrary positions of the input tape seems to be essential for separating two-dimensional automata with markers, but possibly deterministic and nondeterministic rebound automata with markers can be separated as well.

Another open question is, whether one-dimensional nondeterministic two-way counter automata are able to simulate nondeterministic rebound automata. For deterministic devices such a simulation is given in [7], but it does not carry over to nondeterministic automata.

Some relations between classes of languages are shown in the following diagram, where 2NC and 2DC denote deterministic and nondeterministic two-way counter languages, NRA and DRA denote the classes of languages accepted by nondeterministic and deterministic rebound automata. An edge between class X above Y indicates $Y \subseteq X$. The separation of 2NC and 2DC is due to Chrobak [2].



Acknowledgments. I wish to thank Professor K. Inoue for references related to rebound automata and Holger Austinat for comments on an earlier draft of this paper. Support by “Deutsche Akademie der Naturforscher Leopoldina”, grant number BMBF-LPD 9901/8-1 of “Bundesministerium für Bildung und Forschung”, is gratefully acknowledged.

References

1. M. Blum and C. Hewitt. Automata on a 2-dimensional tape. In *Proceedings of the 8th Annual Symposium on Switching and Automata Theory, Austin, 1967*, pages 155–160, 1967.
2. M. Chrobak. Variations on the technique of Duris and Galil. *Journal of Computer and System Sciences*, 30:77–85, 1985.
3. P. Āuriš and Z. Galil. Fooling a two way automaton or one pushdown store is better than one counter for two way machines. *Theoretical Computer Science*, 21:39–53, 1982.
4. P. C. Fischer, A. R. Meyer, and A. L. Rosenberg. Counter machines and counter languages. *Mathematical Systems Theory*, 2:265–283, 1968.
5. K. Inoue and I. Takanami. A survey of two-dimensional automata theory. *Information Sciences*, 55:99–121, 1991.
6. K. Inoue, I. Takanami, and H. Taniguchi. A note on rebound automata. *Information Sciences*, 26:87–93, 1982.
7. K. Morita, K. Sugata, and H. Umeo. Computation complexity of n -bounded counter automaton and multidimensional rebound automaton. *Systems • Computers • Controls*, 8:80–87, 1977. Translated from Denshi Tsushin Gakkai Ronbunshi (IECE of Japan Trans.) 60-D:283–290, 1977 (Japanese).
8. K. Morita, K. Sugata, and H. Umeo. Computational complexity of n -bounded counter automaton and multi-dimensional rebound automaton. *IECE of Japan Trans.*, 60-E:226–227, 1977. Abstract of [7].
9. H. Petersen. Fooling rebound automata. In M. Kutylowski, L. Pacholski, and T. Wierzbicki, editors, *Proceedings of the 24th Symposium on Mathematical Foundations of Computer Science (MFCS), Szklarska Poreba, 1999*, number 1672 in Lecture Notes in Computer Science, pages 241–250, Berlin-Heidelberg-New York, 1999. Springer.
10. A. Rosenfeld. *Picture Languages*. Academic Press, New York, 1979.
11. M. Sakamoto, K. Inoue, and I. Takanami. A two-way nondeterministic one-counter language not accepted by nondeterministic rebound automata. *IECE of Japan Trans.*, 73-E:879–881, 1990.
12. K. Sugata, H. Umeo, and K. Morita. The language accepted by a rebound automaton and its computing ability. *Electronics and Communications in Japan*, 60-A:11–18, 1977.
13. L. Zhang, T. Okozaki, K. Inoue, A. Ito, and Y. Wang. A note on probabilistic rebound automata. *IEICE Trans. Inf. & Syst.*, E81-D:1045–1052, 1998.

Unary Pushdown Automata and Auxiliary Space Lower Bounds*

Giovanni Pighizzini

Dipartimento di Scienze dell'Informazione – Università degli Studi di Milano
via Comelico 39 – 20135 Milano, Italy – pighizzi@dsi.unimi.it

Abstract. It is well-known that context-free languages defined over a one letter alphabet are regular. This implies that unary pushdown automata and unary context-free grammars can be transformed into equivalent nondeterministic and deterministic finite automata. In this paper, we state some upper bounds on the number of states of the resulting automata, with respect to the size of the descriptions of the given pushdown automata and context-free grammars. As a main consequence, we are able to prove a $\log \log n$ lower bound for the workspace used by one-way auxiliary pushdown automata in order to accept nonregular unary languages. The notion of space we consider is the so called *weak space*.

1 Introduction

In the theory of formal languages, *unary* or *tally* languages, i.e., languages defined over a one letter alphabet, have been the subject of many interesting and fruitful researches (e.g., [4,6,14]). In a lot of cases, these studies display important differences between the universe of all languages and the world of unary languages. Probably, the first result of this kind is the collapse of the classes of unary context-free and regular languages, proved by Ginsburg and Rice [9]. An immediate consequence is that every pushdown automaton accepting a unary language can be simulated by a finite state automaton.

In this paper, we further deepen this investigation. By carrying on the analysis of the costs, in terms of states, of the simulations between different kinds of unary automata (i.e., automata accepting unary languages) [4,16], we state upper bounds on the number of states of nondeterministic and deterministic finite automata simulating a given unary pushdown automaton. In particular, in Section 3, we observe that for any pushdown automaton with n states and m pushdown symbols, there is an equivalent context-free grammar in Chomsky normal form, with $h = n^2m + 1$ variables. Subsequently, in Sections 4 and 5, we prove that this grammar can be transformed in equivalent nondeterministic and deterministic automata with 2^{2h} and $O(2^{h^2})$ states, respectively.

The interest for these results extends beyond the study of automaton simulations. In fact, they are a fundamental step for the solution, presented in Section 6, of a problem, related to space lower bounds.

* Partially supported by MURST, under the project “Modelli di calcolo innovativi: metodi sintattici e combinatori”.

We recall that the analysis of the minimal amount of space used by Turing machines to accept nonregular languages, started with the fundamental work of Stearns, Lewis and Hartmanis [18], and has been extensively considered in the literature (e.g., [1,2,12]). This investigation is also related to several questions in structural complexity, in particular to sublogarithmic space bounded computations (see, e.g., [7,8,19]).

Some different space notions have been approached during these studies. Among them, we now recall *strong* and *weak space*. A machine M works in *strong* $s(n)$ space if and only if *any computation on each input* of length n uses no more than $s(n)$ worktape cells [12,18]; M works in *weak* $s(n)$ space if and only if, on each accepted input of length n , there exists *at least one accepting computation* using no more than $s(n)$ worktape cells [1]. Of course, if M accepts a language L in strong $s(n)$ space, then it accepts the same language also in weak $s(n)$ space.

The same space notions can be introduced also for *one-way auxiliary pushdown automata*, i.e., pushdown automata extended with a worktape. Considering these devices, the question arises of finding the minimal amount of worktape space, if any, needed to recognize noncontext-free languages. For strong space, this question was solved by Brandenburg [3], by proving a $\log \log n$ lower bound, whose optimality is witnessed by a unary language.¹

The situation in the weak case is very different. In fact, as proved by Chytil [5], for any integer k , there exists a noncontext-free language accepted in space $O(\log^{(k)} n)$.² A crucial point in this result, is that the language L_k is defined over an alphabet of at least two symbols. Up to now, the study of the unary case was left open.

In Section 6, we get a solution to this problem. In particular, we are able to show that the $\log \log n$ lower bound for the strong case, holds also in the weak case for unary languages, and it is optimal. This result is a non trivial consequence of our simulation of unary pushdown automata by deterministic finite automata, presented in Section 5. Furthermore, this result shows another main difference between unary and general languages and between strong and weak space: when we consider the unary case, or strong space, the lower bound is $\log \log n$; instead, in the case of weak space, there are nonunary noncontext-free languages accepted within very slowly growing space bounds.

For brevity reasons, the proofs are omitted or just outlined in this version of the paper.

2 Preliminaries

In this section, we recall basic notions, notations and facts used in the paper. (For more details see, e.g., [13].)

¹ Actually, the space definition presented in [3] corresponds to the weak notion. However, the argument used to prove the lower bound works only for strong space.

² Throughout the paper, $\log^{(k)}$ denotes the *iterated logarithm*, namely, $\log^{(1)} z = \log z$ and $\log^{(k)} z = \log^{(k-1)} \log z$ for $k > 1$.

For any $z > 0$, $\log z$ is the logarithm of z taken to the base 2. The *greatest common divisor* of integers $a_1, \dots, a_s$ writes as $\gcd(a_1, \dots, a_s)$, their *least common multiple* as $\text{lcm}(a_1, \dots, a_s)$. The following result will be useful:

Theorem 1. [4, Cor. B] *Let $a_1, \dots, a_s$ be positive integers $\leq n$, and $X = \{a_1x_1 + \dots a_sx_s \mid x_1, \dots, x_s \geq 0\}$. Then, the set of numbers in X greater than n^2 coincides with the set of multiples of $\gcd(a_1, \dots, a_s)$ greater than n^2 .*

Given an alphabet Σ , Σ^* denotes the set of strings on Σ , with the empty string ϵ , and Σ^+ the set $\Sigma^* - \{\epsilon\}$. Given a string $x \in \Sigma^*$, $|x|$ denotes its length. Given an integer n , by $L^{<n}$ ($L^{\leq n}$, resp.) we denote the set of strings of length less than n (no greater than n , resp.) belonging to the language L . A language L is said to be *unary* (or *tally*) whenever it can be built over a *single letter* alphabet. In this case, we let $L \subseteq a^*$.

In the paper, we consider nondeterministic finite automata (nfa), finite automata with ϵ -moves, and deterministic finite automata (dfa), over the unary alphabet $\{a\}$. Any automaton with ϵ -moves can be simulated by an nfa with the same number of states. Note that the transition graph of a unary dfa M , i.e., of a dfa over the alphabet $\Sigma = \{a\}$, consists of a path, which starts from the initial state, and it is followed by a cycle of one or more states. As in [4], the *size* of M is the pair (λ, μ) , where $\lambda \geq 1$ and $\mu \geq 0$ denote the number of states which belong to the cycle and to the path, respectively. In particular, it is easy to prove that a unary language L is accepted by a dfa of size (λ, μ) if and only if, for any integer $m \geq \mu$, $a^m \in L$ exactly when $a^{m+\lambda} \in L$.

A *context-free grammar* (cfg, for short), is a tuple $G = (V, \Sigma, P, S)$, where V is the set of variables, Σ is the set of terminals, $S \in V$ is the initial symbol and $P \subseteq V \times (V \cup \Sigma)^*$ is the finite set of productions. A production $(A, \alpha) \in P$ is denoted as $A \rightarrow \alpha$. Relations $\Rightarrow$, $\stackrel{*}{\Rightarrow}$, and $\stackrel{+}{\Rightarrow}$ are defined in the usual way. Given $\alpha, \beta \in (V \cup \Sigma)^*$, if θ is a derivation of β from α , then we write $\theta : \alpha \stackrel{*}{\Rightarrow} \beta$. The set of all variables that occur in the sentential forms of θ is denoted as $\nu(\theta)$. The language generated by the grammar G , i.e., the set $\{x \in \Sigma^* \mid S \stackrel{*}{\Rightarrow} x\}$ is denoted by $L(G)$. It is well-known that the class of languages generated by cfg's coincides with the class of context-free languages. For sake of simplicity, we will consider languages without the empty word ϵ . However, our results can be easily extended to languages containing ϵ . The class of context-free languages properly contains that of regular languages, but, in the unary case, these two classes collapse [9].

Context-free languages can be characterized also by *pushdown automata*. As usual, a *pushdown automaton* (pda, for short) is denoted as a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where Q is the finite set of states, Σ is the input alphabet, Γ is the pushdown alphabet, $q_0 \in Q$ is the initial state, $Z_0 \in \Gamma$ is the start symbol, $F \subseteq Q$ is the set of final states. Without loss of generality, we make the following assumptions about pda's:

- (i) at the start of the computation the pushdown store contains only the start symbol Z_0 ; this symbol is never pushed or popped on the stack;

- (ii) the input is accepted if and only if the automaton reaches a final state, the pushdown store contains only Z_0 and all the input has been scanned;
- (iii) if the automaton moves the input head, then no operations are performed on the stack;
- (iv) every push adds exactly one symbol on the stack.

Note that the transition function δ of a pda M can be written as

$$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times (\{-, pop\} \cup \{push(A) \mid A \in \Gamma\})}.$$

In particular, for $q, p \in Q$, $A, B \in \Gamma$, $\sigma \in \Sigma$, $(p, -) \in \delta(q, \sigma, A)$ means that the pda M , in the state q , with A at the top of the stack, by consuming the input σ , can reach the state p without changing the stack content; $(p, pop) \in \delta(q, \epsilon, A)$ ($(p, push(B)) \in \delta(q, \epsilon, A)$, $(p, -) \in \delta(q, \epsilon, A)$, respectively) means that M , in the state q , with A at the top of the stack, without reading any input symbol, can reach the state p by popping off the stack the symbol A on the top (by pushing the symbol B on the top of the stack, without changing the stack, respectively).

In order to evaluate the complexities of grammars and finite automata equivalent to a given pda M , we will consider two parameters: the number n of states of M and the number m of pushdown symbols, i.e., the cardinality of the alphabet Γ . In fact, for a fixed input alphabet Σ , each pda satisfying the above condition (iv) has a description whose length is polynomial in n and m . Without this condition, other parameters, as the maximum number of symbols that can be pushed on the stack in one move, have to be considered. For instance, note that, for each $n \geq 1$, the regular language $L_n = \{a^{kn} \mid k \geq 0\}$, that requires n states to be recognized by an nfa or by a dfa, can be accepted by a pda with 2 states and 2 pushdown symbols that, in one move, is able to push n symbols on the stack. Similar considerations can be formulated for grammars. The language L_n is generated by the grammar containing only one variable S and the productions $S \rightarrow a^n$ and $S \rightarrow a^n S$. However, it is not difficult to see that for grammars in Chomsky normal form, the number of variables is a “reasonable” measure of complexity [11].

We recall that a *one-way nondeterministic auxiliary pushdown automaton* (auxpda, for short) is a pushdown automaton, extended with a read/write worktape. At the start of the computation the worktape is empty. Moves are defined as for pda's, with the following differences: each transition depends even on the content of the currently scanned worktape cell; a transition modifies also the worktape, by writing a new symbol on the currently scanned cell and by moving the worktape head one position left or right.

Also for auxpda's, without loss of generality, we make assumptions (i), (ii), (iii), and (iv). Space complexity is defined considering only the auxiliary worktape.

3 Pushdown Automata and Context-Free Grammars

In this section, we study the reduction of pda's to cfg's in Chomsky normal form. By slightly modifying standard techniques, we show how to get, from a given

pda $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, with n states and m pushdown symbols, an equivalent context-free grammar in Chomsky normal form with $n^2m + 1$ variables. First, from M , we define the grammar $G_1 = (V, \Sigma, P_1, S)$, where elements of V are triples $[q, A, p]$, with $q, p \in Q$, $A \in \Gamma$, plus the start symbol S , and P_1 contains the following productions:

1. $[q, A, p] \rightarrow [q, A, r][r, A, p]$, for $q, p, r \in Q$, $A \in \Gamma$;
2. $[q, A, p] \rightarrow [q', B, p']$, for $q, q', p, p' \in Q$, $A, B \in \Gamma$ such that $(q', \text{push}(B)) \in \delta(q, \epsilon, A)$ and $(p, \text{pop}) \in \delta(p', \epsilon, B)$;
3. $[q, A, p] \rightarrow \sigma$, for $q, p \in Q$, $\sigma \in \Sigma \cup \{\epsilon\}$, $A \in \Gamma$ such that $(p, -) \in \delta(q, \sigma, A)$;
4. $[q, A, q] \rightarrow \epsilon$, for $q \in Q$, $A \in \Gamma$;
5. $S \rightarrow [q_0, Z_0, q]$, for $q \in F$.

It is possible to prove the following result:

Lemma 1. *For any $x \in \Sigma^*$, $q, p \in Q$, $A \in \Gamma$, $[q, A, p] \xRightarrow{*} x$ if and only if there exists a computation $\mathcal{C}$ of M verifying the following conditions:*

- (i) $\mathcal{C}$ starts in the state q and ends in the state p ; in both these moments the symbol at the top of the stack is A and the height of the stack is the same;
- (ii) during $\mathcal{C}$ the stack is never popped under its level at the beginning of $\mathcal{C}$;
- (iii) the input factor consumed during $\mathcal{C}$ is x .

We point out that the productions in the above item 3 (4, respectively), describe the computations $\mathcal{C}$ of M satisfying the conditions (i) and (ii) of Lemma 1, and consisting of exactly one step (zero steps, respectively).

As a consequence of Lemma 1, it is easy to show that the language generated by G_1 coincides with the language accepted by the given pda M . Furthermore, by applying standard techniques (as described, for instance, in [13]), it is possible to eliminate from G_1 all unit and ϵ -productions, in order to obtain an equivalent grammar G in Chomsky normal form, with the *same* set of variables. Hence:

Theorem 2. *For any pda, with n states and m pushdown symbols, there exists an equivalent cfg in Chomsky normal form, with $n^2m + 1$ variables.*

Further results comparing the descriptonal complexities of pushdown automata and of context-free grammars can be found in [10].

4 Simulation by Nondeterministic Automata

In this section, we study the simulation of unary cfg's by nfa's. In particular, we show that for any given unary cfg $G = (V, \{a\}, P, S)$ in Chomsky normal form with h variables, there exists an equivalent nfa with 2^{2h} states.

Let us start with the following preliminary result, whose proof can be done by using standard arguments related to derivation trees of cfg's:

Lemma 2. *Given the grammar G with h variables, let $\theta : S \xRightarrow{*} a^l$ be a derivation.*

- (i) For each variable $A \in \nu(\theta)$ and for each derivation $\theta_1 : A \xRightarrow{\pm} a^i Aa^j$, there exists a derivation $\theta' : S \xRightarrow{*} a^{l+i+j}$, with $\nu(\theta') = \nu(\theta) \cup \nu(\theta_1)$.
- (ii) For each integer $l \geq 2^h$ there exist three integers s, i, j , with $l = s + i + j$, $s > 0$, and $0 < i + j < 2^h$, a derivation $\theta_1 : S \xRightarrow{*} a^s$, a variable $A \in \nu(\theta_1)$, and a derivation $\theta_2 : A \xRightarrow{\pm} a^i Aa^j$, such that $\nu(\theta) = \nu(\theta_1) \cup \nu(\theta_2)$.

As a consequence of Lemma 2, the language $L(G)$ coincides with the set of strings that can be generated by the following nondeterministic procedure:

```

nondeterministically select a derivation  $\theta : S \xRightarrow{*} a^l$ , with  $l < 2^h$ 
enabled  $\leftarrow \nu(\theta)$ 
iterate  $\leftarrow$  nondeterministically choose true or false
while iterate do
  nondeterministically select a derivation  $\theta_1 : A \xRightarrow{\pm} a^i Aa^j$ ,
    with  $0 < i + j < 2^h$  and  $A \in \text{enabled}$ 
  enabled  $\leftarrow \text{enabled} \cup \nu(\theta_1)$ 
   $l \leftarrow l + i + j$ 
  iterate  $\leftarrow$  nondeterministically choose true or false
endwhile
output  $a^l$ 

```

Note that, at the beginning of each iteration, the variable *enabled* contains the set of non terminals which occur in the derivation so far simulated. One of them is used to pump the output string. We now describe an automaton $M = (Q, \delta, q_0, F)$, with ϵ -moves, implementing a similar strategy. In particular, each state of M is defined by two components: a set of enabled variables and an integer, used to count input factors. Formally, M is defined as follows:

- $Q = 2^V \times \{0, \dots, 2^h - 1\}$;
- $q_0 = (\emptyset, 0)$;
- for $\alpha \subseteq V$, $0 \leq l \leq 2^h - 1$: $\delta((\alpha, l), a) = \begin{cases} \{(\alpha, l+1)\} & \text{if } l < 2^h - 1 \\ \emptyset & \text{otherwise,} \end{cases}$ and,
- $\delta((\alpha, l), \epsilon) = \begin{cases} \{(\beta, 0) \mid \exists \theta : S \xRightarrow{*} a^l \text{ and } \nu(\theta) = \beta\} & \text{if } \alpha = \emptyset \\ \{(\beta, 0) \mid \exists A \in \alpha \exists \theta : A \xRightarrow{\pm} a^i Aa^j, \text{ s.t.} \\ \quad i + j = l \text{ and } \beta = \alpha \cup \nu(\theta)\} & \text{otherwise;} \end{cases}$
- $F = \{(\alpha, 0) \mid \alpha \neq \emptyset\}$.

The main property of the automaton M , useful to prove that the language accepted by it coincides with $L(G)$, is the following:

Lemma 3. *Given $x \in \Sigma^+$ and $\alpha \subseteq V$, $(\alpha, 0) \in \delta((\emptyset, 0), x)$ if and only if there exists a derivation $\theta : S \xRightarrow{*} x$ such that $\nu(\theta) = \alpha$.*

Since ϵ -moves can be eliminated from nondeterministic automata, without increasing the number of states, as a consequence of Lemma 3 and of Theorem 2, we are able to get the following results:

Corollary 1. *For any unary cfg in Chomsky normal form with h variables, there exists an equivalent nfa with at most 2^{2h} states.*

Corollary 2. *For any unary pda with n states and m pushdown symbols, there exists an equivalent nfa with at most 2^{2n^2m+2} states.*

We point out that, as observed in Section 2, the results of Corollary 1 and of Corollary 2 do not hold if we consider general cfg's of general pda's.

5 Simulation by Deterministic Automata

By Corollary 1, given a unary cfg $G = (V, \{a\}, P, S)$ in Chomsky normal form with h variables, there exists an equivalent nfa with at most 2^{2h} states. This automaton can be transformed in a dfa applying the subset construction or the determinization procedure for unary automata, presented in [4]. In both cases, the number of states of the resulting dfa is bounded by a function which grows at least as a double exponential in h .

In this section, we prove that this cost can be dramatically reduced. In fact, we show that there exists a dfa, equivalent to G , with at most $O(2^{h^2})$ states.

Let L denote the language generated by the given grammar G . For any $A \in V$, let L_A be the set of strings in L having a derivation which uses the variable A , i.e., $L_A = \{x \in \Sigma^* \mid \exists \theta : S \xrightarrow{+} x \text{ with } A \in \nu(\theta)\}$. Of course, $L = L_S$.

We say that a variable $A \in V$ is *cyclic* whenever there are two integers i and j such that $A \xrightarrow{+} a^i A a^j$. In order to state the main result of this section, we now evaluate, for any cyclic variable A , the size of a dfa accepting L_A .

Lemma 4. *Given a cyclic variable $A \in V$, and an integer λ , with $0 < \lambda < 2^h$, such that $A \xrightarrow{+} a^i A a^j$ and $i + j = \lambda$, the language L_A is accepted by a dfa of size $(\lambda, h2^h + 2^{2h})$, i.e., for any integer $x \geq h2^h + 2^{2h}$, $a^x \in L_A$ if and only if $a^{x+\lambda} \in L_A$.*

Proof. (outline) To show that L_A is regular, it is enough to observe that it is accepted by the automaton $M_A = (Q, \delta, q_0, F_A)$, where Q, δ, q_0 are defined as for the automaton M presented in Section 4, while $F_A = \{(\alpha, 0) \mid A \in \alpha\}$.

For any integer $x \geq 0$, by Lemma 2(i), $a^x \in L_A$ implies $a^{x+\lambda} \in L_A$.

Conversely, consider $x \geq h2^h + 2^{2h}$ such that $a^{x+\lambda} \in L_A$, and a computation path $\mathcal{C}$ in M_A accepting $a^{x+\lambda}$. Let $l_1, \dots, l_s$ be the lengths of the input factors consumed in the simple cycles of M_A containing states of the path $\mathcal{C}$, i.e.,

$$\{l_1, \dots, l_s\} = \{i > 0 \mid \exists \alpha \subseteq V, (\alpha, i) \text{ is a state of } \mathcal{C} \text{ and } (\alpha, 0) \in \delta((\alpha, i), \epsilon)\}.$$

Let $l = \gcd(\lambda, l_1, \dots, l_s)$. Since $a^{x+\lambda} \in L_A$, $x + \lambda = x_0 + l_1 x_1 + \dots + l_s x_s$, where x_0 is the number of input symbols consumed in the simple path $\mathcal{C}_0$ obtained by deleting from $\mathcal{C}$ the cycles, and $x_i \geq 0$, $i = 1, \dots, s$, is the number of times some cycle consuming l_i symbols is entered in $\mathcal{C}$.

Note that $x_0 \leq (h+1)(2^h-1)$. In fact, for $\alpha, \beta \subseteq V$ and $0 \leq i \leq 2^h-1$, $(\beta, 0) \in \delta((\alpha, i), \epsilon)$ implies $\alpha \subseteq \beta$. Hence, the number of different first components of the states in $\mathcal{C}_0$, is at most $h+1$. Moreover, between two subsequent ϵ -moves, M_A can consume at most $2^h - 1$ input symbols.

The number $l_1x_1 + \dots + l_sx_s$ is a multiple of $\gcd(l_1, \dots, l_s)$, and, then, of l . This implies the existence of an integer k such that $l_1x_1 + \dots + l_sx_s = kl$. Moreover, $x - x_0 = kl - \lambda \geq 2^h(2^h - 1)$ and $x - x_0$ is a multiple of l . Since λ and all l_i 's are less than 2^h , by Theorem 1, there are integers $y_0, y_1, \dots, y_s \geq 0$ such that $x - x_0 = \lambda y_0 + l_1y_1 + \dots + l_sy_s$. Hence, the simple path C_0 can be padded, by inserting y_i times a cycle consuming l_i input symbols ($i = 1, \dots, s$), in order to obtain a new path accepting $a^{x_0+l_1x_1+\dots+l_sx_s} = a^{x-\lambda y_0}$. Furthermore, since we have already proved that, for $z \geq 0$, $a^z \in L$ implies $a^{z+\lambda} \in L$, from $a^{x-\lambda y_0} \in L_A$, it follows that $a^x \in L_A$. $\square$

At this point, we are able to prove the main result of this section:

Theorem 3. *For any unary cfg in Chomsky normal form with h variables there exists an equivalent dfa with $O(2^{h^2})$ states.³*

Proof. (outline) We can express the language $L = L(G)$ as:

$$L = L^{<2^h} \cup \bigcup_{A \in V \text{ s.t. } A \text{ is cyclic}} L_A.$$

The language $L^{<2^h}$ is finite. It is easy to show that it can be accepted by a dfa of size $(1, 2^h)$. Furthermore, by Lemma 4, for any cyclic variable A , the language L_A can be accepted by a dfa of size $(\lambda_A, h2^h + 2^{2h})$, with $\lambda_A < 2^h$.

It is not difficult to show that if a unary regular language $\tilde{L}$ is the union of k languages, which are accepted by dfa's of size $(\lambda_1, \mu_1), \dots, (\lambda_k, \mu_k)$, respectively, then $\tilde{L}$ is accepted by a dfa of size $(\text{lcm}(\lambda_1, \dots, \lambda_k), \max(\mu_1, \dots, \mu_k))$. Hence, the given language L is accepted by a dfa of size (λ, μ) , where $\lambda = \text{lcm}(\{\lambda_A \mid A \text{ is cyclic}\}) < (2^h)^h = 2^{h^2}$, and $\mu = h2^h + 2^{2h}$. $\square$

We point out that, since L coincides with L_S , when the variable S is cyclic the period of L coincides with λ_S . In this case the number of states in the cyclic part of the dfa accepting L is bounded by 2^h and the total number of states of the resulting dfa is $2^{O(h)}$. From Theorem 2 and Theorem 3, we get:

Corollary 3. *For any unary pda with n states and m pushdown symbols, there exists an equivalent dfa with $O(2^{n^4m^2})$ states.*

6 An Optimal Lower Bound for Unary AuxPda's

In this section, we study unary auxiliary pushdown automata working in *weak* space. By using the simulation result of Corollary 3, we are able to prove that $\log \log n$ is the minimal amount of space needed by these devices in order to recognize unary noncontext-free languages. Furthermore, this lower bound is

³ After the submission of this paper, we learned that the result stated in Theorem 3 was independently obtained by Ming-Wei Wang and Jeffrey Shallit. They also proved that it is tight. These results will be collected in a future joint paper.

optimal. We point out that, when the input alphabet contains at least two symbols, the situation is very different: as shown in [5], for any integer k there exists a noncontext-free language accepted in weak $O(\log^{(k)} n)$ space.

To state our result, it is useful to recall the notion of *automaticity* [17], which is a measure of the complexity of the description of a language by deterministic automata. In particular, the automaticity of a regular language is a constant, while the automaticity of a nonregular language grows at least as a linear function. More precisely:

Definition 1. *Given a language $L \subseteq \Sigma^*$, the automaticity of L is the function $A_L : \mathbf{N} \rightarrow \mathbf{N}$, which associates with every integer n the minimum number of states of a dfa accepting a language L' such that $L^{\leq n} = L'^{\leq n}$.*

Theorem 4. [15] *Let $L \subseteq \Sigma^*$ be a nonregular language. Then $A_L(n) \geq (n + 3)/2$, for infinitely many n .*

The following result is useful to evaluate the automaticity of unary languages accepted by auxpda's:

Theorem 5. *Given an auxpda M accepting a unary language L in weak $s(n)$ space, for any integer $n \geq 0$ there exists a dfa M_n with the following properties:*

- (i) *the language L_n accepted by M_n coincides with L on strings of length at most n , i.e., $L_n^{\leq n} = L^{\leq n}$;*
- (ii) *the number of states of M_n is bounded by $2^{2^{O(s(n))}}$.*

Proof. (outline) Given the auxpda M and an integer n , we consider the pda M'_n which is obtained from M by encoding, in each state, a state of M and the content of the first $s(n)$ worktape cells. M'_n performs a simulation step by step of M . When the simulated computation of M tries to exceed the first $s(n)$ worktape cells, M'_n stops and rejects.

Since we are considering *weak* space, an input of length n is accepted by M if and only if there is an accepting computation which does not use more than $s(n)$ worktape cells. By construction, the pda M'_n is able to mimic the same accepting computation. Furthermore, to each rejecting computation of M corresponds a rejecting computation of M'_n . Thus, we can easily conclude that $L_n^{\leq n} = L^{\leq n}$.

The number of states of M'_n is $2^{O(s(n))}$. By Corollary 3, there exists a dfa M_n with $2^{2^{O(s(n))}}$ states equivalent to M'_n . $\square$

Now, we are able to prove the main result of this section:

Theorem 6. *Let M be a unary auxpda accepting a noncontext-free language L in weak $s(n)$ space. Then $s(n) \notin o(\log \log n)$.*

Proof. By Theorem 5, given M , there exists a constant k such that, for any integer n , from M it is possible to get a dfa M_n , with at most $2^{2^{ks(n)}}$ states, such that the language L_n accepted by M_n verifies the equality $L^{\leq n} = L_n^{\leq n}$. Hence, the automaticity $A_L(n)$ is bounded by $2^{2^{ks(n)}}$.

Suppose that $s(n) \in o(\log \log n)$. Then $2^{2^{ks(n)}} < (n+3)/2$, for any sufficiently large n . Since L is nonregular, this is a contradiction to Theorem 4. $\square$

We point out that the optimality of the lower bound stated in Theorem 6 is witnessed by the language $L = \{a^{2^i} \mid i \geq 0\}$ that is accepted by a deterministic auxpda in strong (and weak) $O(\log \log n)$ space [3,5].

References

1. M. Alberts. Space complexity of alternating Turing machines. In *Proc. FCT '85*, Lecture Notes in Computer Science 199, pages 1–7. Springer Verlag, 1985.
2. A. Bertoni, C. Mereghetti, and G. Pighizzini. Strong optimal lower bounds for Turing machines that accept nonregular languages. In *Proc. MFCS '95*, Lecture Notes in Computer Science 969, pages 309–318. Springer Verlag, 1995.
3. F. Brandenburg. On one-way auxiliary pushdown automata. In *Proc. 3rd GI Conference*, Lecture Notes in Computer Science 48, pages 133–144, 1977.
4. M. Chrobak. Finite automata and unary languages. *Theoretical Computer Science*, 47:149–158, 1986.
5. M. Chytil. Almost context-free languages. *Fundamenta Inform.*, IX:283–322, 1986.
6. V. Geffert. Tally version of the Savitch and Immerman–Szelepcsényi theorems for sublogarithmic space. *SIAM J. Computing*, 22:102–113, 1993.
7. V. Geffert. Bridging across the $\log(n)$ space frontier. In *Proc. MFCS '95*, Lecture Notes in Computer Science 969, pages 50–65. Springer Verlag, 1995. To appear in *Information and Computation*.
8. V. Geffert, C. Mereghetti, and G. Pighizzini. Sublogarithmic bounds on space and reversals. *SIAM J. Computing*, 28(1):325–340, 1999.
9. S. Ginsburg and H. Rice. Two families of languages related to ALGOL. *Journal of the ACM*, 9(3):350–371, 1962.
10. J. Goldstine, J. Price, and D. Wotschke. A pushdown automaton or a context-free grammar — which is more economical? *Theoretical Comp. Sc.*, 18:33–40, 1982.
11. J. Gruska. Descriptive complexity of context-free languages. In *Proc. MFCS '73*, pages 71–83. Mathematical Institute of the Slovak Academy of Sciences, 1973.
12. J. Hopcroft and J. Ullman. Some results on tape-bounded Turing machines. *Journal of the ACM*, 16:168–177, 1969.
13. J. Hopcroft and J. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, Reading, MA, 1979.
14. J. Kaneps. Regularity of one-letter languages acceptable by 2-way finite probabilistic automata. In *Proc. FCT '91*, Lecture Notes in Computer Science 529, pages 287–296. Springer Verlag, 1991.
15. R.M. Karp. Some bounds on the storage requirements of sequential machines and Turing machines. *Journal of the ACM*, 14(3):478–489, 1967.
16. C. Mereghetti and G. Pighizzini. Optimal simulations between unary automata. In *Proc. STACS '98*, Lecture Notes in Computer Science 1373, pages 139–149. Springer, 1998.
17. J. Shallit and Y. Breitbart. Automaticity I: Properties of a measure of descriptive complexity. *J. Comp. and System Sciences*, 53(1):10–25, 1996.
18. R. Stearns, J. Hartmanis, and P. Lewis. Hierarchies of memory limited computations. In *IEEE Conf. Record on Switching Circuit Theory and Logical Design*, pages 179–190, 1965.
19. A. Szepietowski. *Turing Machines with Sublogarithmic Space*. Lecture Notes in Computer Science 843. Springer Verlag, 1994.

Binary Decision Diagrams by Shared Rewriting

Jaco van de Pol^{1*} and Hans Zantema^{1,2**}

¹ CWI, P.O.-box 94.079, 1090 GB Amsterdam, The Netherlands

² Department of Computer Science, Utrecht University
P.O.-box 80.089, 3508 TB Utrecht, The Netherlands

Abstract. In this paper we propose a uniform description of basic BDD theory and algorithms by means of term rewriting. Since a BDD is a DAG instead of a tree we need a notion of shared rewriting and develop appropriate theory. A rewriting system is presented by which canonical forms can be obtained. Various reduction strategies give rise to different algorithms. A *layerwise* strategy is proposed having the same time complexity as the traditional *apply*-algorithm, and the *lazy* strategy is studied, which resembles the existing *up-one*-algorithm. We show that these algorithms have incomparable performance.

1 Introduction

Equivalence checking and satisfiability testing of propositional formulas are basic but hard problems in many applications, including hardware verification [4] and symbolic model checking [5]. Binary decision diagrams (BDDs) [2,3,8] are an established technique for this kind of boolean formula manipulation. The basic ingredient is representing a boolean formula by a unique canonical form, the so called reduced ordered BDD (ROBDD). After canonical forms have been established equivalence checking and satisfiability testing are trivial. Constructing the canonical form however, can be exponential.

Various extensions to the basic data-type have been proposed, like DDDs [9], BEDs [1] and EQ-BDDs [6]. Many variants of Bryant's original *apply*-algorithm for computing boolean combinations of ROBDDs have been proposed in the literature. Usually, such adaptations are motivated by particular benchmarks, that show a speed-up for certain cases. In many cases, the relative complexity between the variants is not clear and difficult to establish due to the variety of data-types.

Therefore, we propose to use term rewriting systems (TRS) as a uniform model for the study of operations on BDDs. By enriching the signature, extended data types can be modeled. Various different algorithms can be obtained from a fixed TRS by choosing a reduction *strategy*. In our view, this opens the way in which the BDD-world can benefit from the huge amount of research on rewriting strategies (see [7] for an overview).

* Email: Jaco.van.de.Pol@cwi.nl

** Email: hansz@cs.uu.nl

A complication is that the relative efficiency of BDDs hinges on the maximally shared representation. In Section 2 we present an elegant abstraction of maximally shared graph rewriting, in order to avoid its intricacies. Instead of introducing a rewrite relation on graphs, we introduce a *shared rewrite step* on terms. In a shared rewrite step, all identical redexes have to be rewritten at once. We prove that if a TRS is terminating and confluent, then the shared version is so too. This enables us to lift rewrite results from standard term rewriting to the shared setting for free.

In Section 3, we present a TRS for applying logical operations to ROBDDs and prove its correctness. Because a TRS-computation is non-deterministic, this proves the correctness of a whole class of algorithms. In particular, we reconstruct the traditional *apply*-algorithm as an application of the so-called *layerwise* strategy. We also investigate the well-known *innermost* and *lazy* strategies. The lazy strategy happens to coincide with the *up-one* algorithm in [1] (those authors argue that their *up-all* algorithm is similar to the traditional *apply*).

Finally we provide series of examples to show that the innermost strategy performs quite bad, and that the *apply*-algorithm and the lazy strategy have incomparable complexity. In [1] an example is given for one direction, but this depends on additional structural rules. An extended version of this paper appeared as [11].

2 Shared Term Rewriting

We assume familiarity with standard notions from term rewriting. See [7] for an introduction. The size of a term T is usually measured as the number of its internal nodes, viewed as a *tree*. This is inductively defined as $\#(T) = 0$ if T is a constant or a variable, and $\#(f(T_1, \dots, T_n)) = 1 + \#(T_1) + \dots + \#(T_n)$.

However, for efficiency reasons, most implementations apply the *sharing* technique. Each subterm is stored at a certain location in the memory of the machine, various occurrences of the same subterm are replaced by a pointer to this single location. This shared representation can be seen as a directed acyclic graph (DAG). Mathematically, we define the *maximally* shared representation of a term as the set of its subterms. It is clear that there is a one-to-one correspondence between a tree and its maximally shared representation.

A natural size of the shared representation is the number of nodes in the DAG. So we define the shared size of a term:

$$\#_{sh}(t) = \#\{s \mid s \text{ is a subterm of } t\}.$$

The size of the shared representation can be much smaller than the tree size as illustrated by the next example, which is exactly the reason that sharing is applied.

Example 1. Define $T_0 = \text{true}$ and $U_0 = \text{false}$. For binary symbols $p_1, p_2, p_3, \dots$ define inductively $T_n = p_n(T_{n-1}, U_{n-1})$ and $U_n = p_n(U_{n-1}, T_{n-1})$. Considering T_n as a term its size $\#(T_n)$ is exponential in n . However, the only subterms of T_n are *true*, *false*, and T_i and U_i for $i < n$, hence $\#_{sh}(T_n)$ is linear in n . $\square$

Maximal sharing is essentially the same as what is called the *fully collapsed tree* in [10]. In implementations some care has to be taken in order to keep terms maximally shared. In essence, when constructing or modifying a term, a hash table is used to find out whether a node representing this term exists already. If so, this node is reused; otherwise a new node is created. In order to avoid these difficulties in complexity analysis, we introduce the *shared rewrite relation* $\Rightarrow$ on terms. In a shared rewrite step, all occurrences of a redex have to be rewritten at once. We will take the maximum number of $\Rightarrow$ -steps from t as the time complexity of computing t .

Definition 1. *For terms t and t' there is a shared rewrite step $t \Rightarrow_R t'$ with respect to a rewrite system R if $t = C[l^\sigma, \dots, l^\sigma]$ and $t' = C[r^\sigma, \dots, r^\sigma]$ for one rewrite rule $l \rightarrow r$ in R , some substitution σ and some multi-hole context C having at least one hole, and such that l^σ is not a subterm of C .*

Both in unshared rewrite steps $\rightarrow_R$ and shared rewrite steps $\Rightarrow_R$ the subscript R is often omitted if no confusion is caused. We now study some properties of the rewrite relation $\Rightarrow_R$. The following lemmas are straightforward from the definition.

Lemma 1. *If $t \Rightarrow t'$ then $t \rightarrow^+ t'$.*

Lemma 2. *If $t \rightarrow t'$ then a term t'' exists satisfying $t' \rightarrow^* t''$ and $t \Rightarrow t''$.*

The next theorem shows how the basic rewriting properties are preserved by sharing. In particular, if $\rightarrow$ is terminating and all critical pairs converge, then termination and confluence of $\Rightarrow$ can be concluded too.

Theorem 1. (1) *If $\rightarrow$ is terminating then $\Rightarrow$ is terminating too.*

(2) *A term is a normal form with respect to $\Rightarrow$ if and only if it is a normal form with respect to $\rightarrow$.*

(3) *If $\Rightarrow$ is weakly normalizing and $\rightarrow$ has unique normal forms, then $\Rightarrow$ is confluent.*

(4) *If $\rightarrow$ is confluent and terminating then $\Rightarrow$ is confluent and terminating too.*

Proof. Part (1) follows directly from Lemma 1.

If t is a normal form with respect to $\rightarrow$ then it is a normal form with respect to $\Rightarrow$ by Lemma 1. If t is a normal form with respect to $\Rightarrow$ then it is a normal form with respect to $\rightarrow$ by Lemma 2. Hence we have proved part (2).

For part (3) assume $s \Rightarrow^* s_1$ and $s \Rightarrow^* s_2$. Since $\Rightarrow$ is weakly normalizing there are normal forms n_1 and n_2 with respect to $\Rightarrow$ satisfying $s_i \Rightarrow^* n_i$ for $i = 1, 2$. By part (2) n_1 and n_2 are normal forms with respect to $\rightarrow$; by Lemma 1 we have $s \rightarrow^* n_i$ for $i = 1, 2$. Since $\rightarrow$ has unique normal forms we conclude $n_1 = n_2$. Since $s_i \Rightarrow^* n_i$ for $i = 1, 2$ we proved that $\Rightarrow$ is confluent.

Part (4) is immediate from part (1) and part (3). $\square$

Note that Theorem 1 holds for any two abstract reduction systems $\rightarrow$ and $\Rightarrow$ satisfying Lemmas 1 and 2 since the proof does not use anything else.

Example 2. (Due to Vincent van Oostrom) The converse of Theorem 1.1 doesn't hold. The rewrite system consisting of the two rules $f(0, 1) \rightarrow f(1, 1)$ and $1 \rightarrow 0$ admits an infinite reduction $f(0, 1) \rightarrow f(1, 1) \rightarrow f(0, 1) \rightarrow \dots$, but the shared rewrite relation $\Rightarrow$ is terminating.

For preservation of confluence the combination with termination is essential, as is shown by the rewrite system consisting of the two rules $0 \rightarrow f(0, 1)$ and $1 \rightarrow f(0, 1)$. This system is confluent since it is orthogonal, but $\Rightarrow$ is not even locally confluent since $f(0, 1)$ reduces to both $f(0, f(0, 1))$ and $f(f(0, 1), 1)$, not having a common $\Rightarrow$ -reduct. $\square$

Notions on reduction strategies like innermost and outermost rewriting carry over to shared rewriting as follows. As usual a redex is defined to be a subterm of the shape l^σ where $l \rightarrow r$ is a rewrite rule and σ is a substitution. A (non-deterministic) reduction strategy is a function that maps every term that is not in normal form to a non-empty set of its redexes, being the redexes that are allowed to be reduced. For instance, in the innermost strategy the set of redexes is chosen for which no proper subterm is a redex itself. This naturally extends to shared rewriting: choose a redex in the set of allowed redexes, and reduce all occurrences of that redex. Note that it can happen that some of these occurrences are not in the set of allowed redexes. For instance, for the two rules $f(x) \rightarrow x$, $a \rightarrow b$ the shared reduction step $g(a, f(a)) \Rightarrow g(b, f(b))$ is an outermost reduction, while only one of the two occurrences of the redex a is outermost.

3 ROBDD Algorithms as Reduction Strategies

We consider a set A of binary *atoms*, whose typical elements are denoted by $p, q, r, \dots$. A *binary decision tree* over A is a binary tree in which every internal node is labeled by an atom and every leaf is labeled either **true** or **false**. In other words, a decision tree over A is defined to be a ground term over the signature having **true** and **false** as constants and elements of A as binary symbols.

Given an instance $s : A \rightarrow \{\mathbf{true}, \mathbf{false}\}$, every decision tree can be evaluated to either **true** or **false**, by interpreting $p(T, U)$ as “if $s(p)$ then T else U ”. So a decision tree represents a boolean function. Conversely, it is not difficult to see that every boolean function on A can be described by a decision tree. One way to do so is building a decision tree such that in every path from the root to a leaf every $p \in A$ occurs exactly once, and plugging the values **true** and **false** in the $2^{\#A}$ leaves according to the $2^{\#A}$ lines of the truth table of the given boolean function. Two decision trees T and U are called *equivalent* if they represent the same boolean function.

A decision tree is said to be in *canonical form* with respect to some total order $<$ on A if on every path from the root to a leaf the atoms occur in strictly increasing order, and no subterm of the shape $p(T_1, T_2)$ exists for which T_1 and T_2 are syntactically equal. A BDD (binary decision diagram) is defined to be a decision tree in which sharing is allowed. An ROBDD (reduced ordered binary decision diagram) can now simply be defined as the maximally shared representation of a decision tree in canonical form.

Theorem 2 (Bryant [2]). *Let $<$ be a total order on A . Then every boolean function can uniquely be represented by an ROBDD with respect to $<$.*

We refer to [11] for our proof of this fact using standard rewriting analysis based on weak normalization and confluence of an appropriate rewrite system, whose normal forms are canonical.

Theorem 2 suggests a way to decide whether two logical formulas are equivalent: bring both expressions to ROBDD form and look whether the results are syntactically equal. We now describe how an arbitrary propositional formula can be transformed to an ROBDD by rewriting. Due to sharing the basic steps of rewriting will be $\Rightarrow$ instead of $\rightarrow$.

As a first step every occurrence of an atom p in the formula is replaced by $p(\text{true}, \text{false})$, being the decision tree in canonical form representing the propositional formula p . The signature of the TRS consists of the constants **true** and **false**, the unary symbol $\neg$, binary symbols for all elements of A and the binary symbols $\vee$, $\wedge$ and **xor**, written infix as usually.

Next we give a rewrite system $\mathcal{B}$ by which the propositional symbols are propagated through the term and eventually removed, reaching the ROBDD as the normal form. In Figure 1, p ranges over A and $\diamond$ ranges over the symbols $\vee$, $\wedge$ and **xor**. The rules of the shape $p(x, x) \rightarrow x$ are called *idempotence rules*, all other rules are called *essential rules*.

$p(x, x) \rightarrow x$	for all p
$\neg p(x, y) \rightarrow p(\neg x, \neg y)$	for all p
$p(x, y) \diamond p(z, w) \rightarrow p(x \diamond z, y \diamond w)$	for all $\diamond, p$
$p(x, y) \diamond q(z, w) \rightarrow p(x \diamond q(z, w), y \diamond q(z, w))$	for all $\diamond, p < q$
$q(x, y) \diamond p(z, w) \rightarrow p(q(x, y) \diamond z, q(x, y) \diamond w)$	for all $\diamond, p < q$
$\neg \text{true} \rightarrow \text{false}$	$\text{true} \wedge x \rightarrow x$
$\neg \text{false} \rightarrow \text{true}$	$x \wedge \text{true} \rightarrow x$
$\text{true} \vee x \rightarrow \text{true}$	$\text{false} \wedge x \rightarrow \text{false}$
$x \vee \text{true} \rightarrow \text{true}$	$x \wedge \text{false} \rightarrow \text{false}$
$\text{false} \vee x \rightarrow x$	$\text{true} \text{ xor } x \rightarrow \neg x$
$x \vee \text{false} \rightarrow x$	$x \text{ xor } \text{true} \rightarrow \neg x$
	$\text{false} \text{ xor } x \rightarrow x$
	$x \text{ xor } \text{false} \rightarrow x$

Fig. 1. The rewrite system $\mathcal{B}$.

We have defined $\mathcal{B}$ in such a way that terms are only rewritten to logically equivalent terms. Hence if a term rewrites in some way by $\mathcal{B}$ to an ROBDD, we may conclude that the result is the unique ROBDD equivalent to the original term (independent of whether the system is confluent).

The rewrite system $\mathcal{B}$ is terminating since every left hand side is greater than the corresponding right hand side with respect to any recursive path order for

a precedence $\succ$ satisfying $\text{xor} \succ \neg \succ b$ and $\diamond \succ p$ for $\diamond \in \{\neg, \vee, \wedge, \text{xor}\}$ and $b \in \{\text{false}, \text{true}\}$ and $p \in A$. Hence reducing will lead to a normal form, and it is easily seen that ground normal forms do not contain symbols $\neg, \vee, \wedge, \text{xor}$. By Theorem 1.(1) this also holds for shared rewriting.

The rewrite system $\mathcal{B}$ is not (ground) confluent, for instance if $q > p$ the term $q(p(\text{false}, \text{true}), p(\text{false}, \text{true})) \wedge q(\text{false}, \text{true})$ reduces to the two distinct normal forms $p(\text{false}, q(\text{false}, \text{true}))$ and $q(\text{false}, p(\text{false}, \text{true}))$. Moreover, we see that $\mathcal{B}$ admits ground normal forms that are not in canonical form. However, when starting with a propositional formula this cannot happen due to the following

Invariant: For every subterm of the shape $p(T, U)$ for $p \in A$ all symbols $q \in A$ occurring in T or U satisfy $p < q$.

In a propositional formula in which every atom p is replaced by $p(\text{true}, \text{false})$ this clearly holds since $T = \text{true}$ and $U = \text{false}$ for every subterm of the shape $p(T, U)$. Further for all rules of $\mathcal{B}$ it is easily checked that if the invariant holds for some term, after application of a $\mathcal{B}$ -rule it remains to hold. Hence for normal forms of propositional formulas the invariant holds. Due to the idempotence rules we now conclude that these normal forms are in canonical form. We have proved the following theorem.

Theorem 3. *Let Φ be a propositional formula over A . Replace every atom $p \in A$ occurring in Φ by $p(\text{true}, \text{false})$ and reduce the resulting term to normal form with respect to $\Rightarrow_{\mathcal{B}}$. Then the resulting normal form is the ROBDD of Φ .*

In this way we have described the process of constructing the unique ROBDD purely by rewriting. Of course this system is inspired by [2,8], but instead of having a deterministic algorithm, we now still have a lot of freedom in choosing the strategy for reducing to normal form. But one strategy may be much more efficient than another. We first show that the leftmost innermost strategy, even when adapted to shared rewriting, may be extremely inefficient.

Example 3. As in Example 1 define $T_0 = \text{true}$ and $U_0 = \text{false}$, and define inductively $T_n = p_n(T_{n-1}, U_{n-1})$ and $U_n = p_n(U_{n-1}, T_{n-1})$.

Both T_n and U_n are in canonical form, hence can be considered as ROBDDs. Both are the ROBDDs of simple propositional formulas, in particular for odd n the term T_n is the ROBDD of $\text{xor}_{i=1}^n p_i$ and U_n of $\neg(\text{xor}_{i=1}^n p_i)$, and for even n the other way around. In fact they describe the *parity* functions yielding **true** if and only if the number of i -s for which p_i holds is even or odd, respectively.

Surprisingly, for every n both for $\neg(T_n)$ and $\neg(U_n) \Rightarrow_{\mathcal{B}}$ -reduction to normal form by the leftmost-innermost strategy requires $2^n - 1$ $\neg$ -steps, where a $\neg$ -step is defined to be an application of a rule $\neg p(x, y) \rightarrow p(\neg x, \neg y)$. We prove this by induction on n . For $n = 0$ it trivially holds. For $n > 0$ the first reduction step is

$$\neg(T_n) \Rightarrow_{\mathcal{B}} p_n(\neg(T_{n-1}), \neg(U_{n-1})).$$

The leftmost-innermost reduction continues by reducing $\neg(T_{n-1})$. During this reduction no $\neg$ -redex is shared in $\neg(U_{n-1})$ since $\neg(U_{n-1})$ contains only one $\neg$ -symbol that is too high in the tree. Hence $\neg(T_{n-1})$ is reduced to normal form

with $2^{n-1} - 1$ $\neg$ -steps due to the induction hypothesis, without affecting the right part $\neg(U_{n-1})$ of the term. After that another $2^{n-1} - 1$ $\neg$ -steps are required to reduce $\neg(U_{n-1})$, making the total of $2^n - 1$ $\neg$ -steps. For $\neg(U_n)$ the argument is similar, concluding the proof.

Although the terms encountered in this reduction are very small in the shared representation, we see that by this strategy every $\Rightarrow$ -step consists of one single $\rightarrow$ -step, of which exponentially many are required. $\square$

We will now show that the standard algorithm based on Bryant's *apply* can essentially be mimicked by a layerwise reduction strategy, having the same complexity. We say that a subterm V of a term T is an *essential redex* if $V = l^\sigma$ for some substitution σ and some essential rule $l \rightarrow r$ in $\mathcal{B}$.

Proposition 1. *Let T, U be ROBDDs.*

- If $\neg T \Rightarrow_{\mathcal{B}}^* V$ then every essential redex in V is of the shape $\neg T'$ for some subterm T' of T .
- If $T \diamond U \Rightarrow_{\mathcal{B}}^* V$ for $\diamond = \vee$ or $\diamond = \wedge$ then every essential redex in V is of the shape $T' \diamond U'$ for some subterm T' of T and some subterm U' of U .
- If $T \text{ xor } U \Rightarrow_{\mathcal{B}}^* V$ then every essential redex in V is of the shape $T' \text{ xor } U'$ or $\neg T'$ or $\neg U'$ for some subterm T' of T and some subterm U' of U .

Proof. This proposition immediately follows from its unshared version: let T, U be decision trees in canonical form and replace $\Rightarrow_{\mathcal{B}}$ in all three assertions by $\rightarrow_{\mathcal{B}}$. This unshared version is proved by induction on the reduction length of $\rightarrow_{\mathcal{B}}^*$ and considering the shape of the rules of $\mathcal{B}$. $\square$

The problem in the exponential leftmost innermost reduction above is that during the reduction very often the same redex is reduced. The key idea now is that in a *layerwise* reduction every essential redex is reduced at most once.

Definition 2. *An essential redex l^σ is called a p -redex for $p \in A$ if p is the smallest symbol occurring in l^σ with respect to $<$. An essential redex l^σ is called an ∞ -redex if no symbol $p \in A$ occurs in l^σ ; define $p < \infty$ for all $p \in A$.*

A redex is called layerwise if either

- *it is a redex with respect to an idempotence rule, or*
- *it is a p -redex for $p \in A \cup \{\infty\}$, and no q -redex for $q < p$ exists, and if the root of the redex is $\neg$ then no p -redex exists of which the root is xor .*

A $\Rightarrow_{\mathcal{B}}$ -reduction is called layerwise if every step consists of the reduction of all occurrences of a layerwise redex.

Clearly every term not in normal form contains a layerwise redex, hence layerwise reduction always leads to the unique normal form. Just like innermost and outermost reduction, layerwise reduction is a non-deterministic reduction strategy. We will show that layerwise reduction leads to normal forms efficiently for suitable terms, due to the following proposition.

Proposition 2. *Let T, U be ROBDDs. In every layerwise $\Rightarrow_{\mathcal{B}}$ -reduction of $\neg T$, $T \vee U$, $T \wedge U$ or $T \text{ xor } U$ every essential redex is reduced at most once.*

Proof. Assume that an essential redex l^σ is reduced twice:

$$C[l^\sigma] \Rightarrow_{\mathcal{B}}^+ C'[l^\sigma] \Rightarrow_{\mathcal{B}} \dots$$

Note that l^σ is a p -redex for some $p \in A \cup \{\infty\}$, because it is essential. Since the reduction is layerwise, every reduction step is either an idempotence step or a reduction of a p -redex for this particular p . Due to Proposition 1 and the shape of the rules the only kind of new p -redexes that can be created in this reduction is a p -redex having $\neg$ as its root, obtained by reducing a p -redex having xor as its root. So this p -redex with root xor already occurs in $C[l^\sigma]$. Since the reduction is layerwise the root of l^σ is not $\neg$. We conclude that the p -redex l^σ in $C'[l^\sigma]$ is not created during this reduction, hence it already occurred in the first term $C[l^\sigma]$. Since we apply shared rewriting this occurrence of l^σ was already reduced in the first step, contradiction. $\square$

Theorem 4. *Let T be an ROBDD. Then every layerwise $\Rightarrow_{\mathcal{B}}$ -reduction of $\neg T$ contains at most $\#_{sh}(T)$ steps.*

Let T, U be ROBDDs. Then every layerwise $\Rightarrow_{\mathcal{B}}$ -reduction of $T \vee U$, $T \wedge U$ or $T \text{ xor } U$ contains $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$ steps.

Proof. If a layerwise reduction of $\neg T$ contains an idempotence step $V \Rightarrow_{\mathcal{B}} V'$, then this idempotence step was also possible on the original term T , contradicting the assumption that T is an ROBDD. Hence a layerwise reduction of $\neg T$ consists only of reductions of essential redexes, and by Proposition 1 the number of candidates is at most $\#_{sh}(T)$. By Proposition 2 each of these possible essential redexes is reduced at most once, hence the total number of steps is at most $\#_{sh}(T)$.

Let V be either $T \vee U$, $T \wedge U$ or $T \text{ xor } U$. Then a layerwise reduction of V consists of a combination of reductions of essential redexes and a number of idempotence steps. By Proposition 1 the number of candidates for essential redexes is $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$, each of which is reduced at most once by Proposition 2. Hence the total number of reductions of essential redexes is $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. Since in every reduction of an essential redex the shared size $\#_{sh}$ increases by at most one, and by every idempotence step $\#_{sh}$ decreases by at least one, the total number of idempotence steps is at most $\#_{sh}(V) + \mathcal{O}(\#_{sh}(T) * \#_{sh}(U)) = \mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. So the total number of steps is $\mathcal{O}(\#_{sh}(T) * \#_{sh}(U))$. $\square$

The procedure sketched above mimics Bryant's original *apply*-function. On formulas with more than one connective, it is repeatedly applied to one of the innermost connectives, thus removing all connectives step by step. It can also be seen as lifting all propositional atoms, for which reason it is called *up-all* in [1]. Note that this is *not* the same as applying the layerwise strategy on the formula itself.

However, other strategies are also conceivable. For instance, we could devise a strategy which brings the smallest atom to the root very quickly. To this end,

we define *head normal forms* to be terms of the form **false**, **true** and $p(T, U)$. The *lazy strategy* is defined to forbid reductions inside T in subterms of the form $T \diamond U$, $U \diamond T$ and $\neg T$ in case T is in head normal form. We will show that the lazy strategy is not comparable to the *apply*-algorithm.

Lemma 3. *Each (unshared) lazy reduction sequence from T leads to a head normal form in at most $2\#(T)$ reduction steps.*

Proof. Induction on T . The cases **false**, **true** and $p(T, U)$ are trivial.

Let $T = P \diamond Q$, with $\diamond \in \{\text{xor}, \wedge, \vee\}$: Let $\#(P) = m$ and $\#(Q) = n$. By induction hypothesis, P reduces to head normal form in at most $2m$ steps. So the lazy strategy allows at most $2m$ reductions in the left hand side of $P \diamond Q$. Similarly, in the right hand side at most $2n$ steps are admitted.

Hence after at most $2(m + n)$ steps, $P \diamond Q$ is reduced to one of: $p(P_1, P_2) \diamond q(Q_1, Q_2)$ or $b \diamond Q_1$ or $P_1 \diamond b$, where $b \in \{\text{false}, \text{true}\}$ and P_i and Q_i are in head normal form for $i = 1, 2$. In most of the cases this reduces to head normal form in the next step, for **true** xor Q_1 and P_1 xor **true** it takes two steps to reach a head normal form. So we use at most $2(m + n) + 2 = 2\#(T)$ steps.

Case $T = \neg P$ is similar but easier. □

Example 4. Let Φ be a formula of size m , whose ROBDD-representation is exponentially large in m (for instance $\bigvee_{i=1}^n (p_i \wedge q_i)$ with $p_i < q_j$ for all i and j [3]). Assume that atom p is smaller than all atoms occurring in formula Φ . Consider the formula $p \wedge (\Phi \wedge \neg p)$, which is clearly unsatisfiable. Note that the traditional algorithm using *apply* will as an intermediate step always completely build the ROBDD for Φ , which is exponential by assumption.

We now show that the lazy strategy has linear time complexity. Replace each atom q by $q(\text{true}, \text{false})$, transforming Φ to Φ' . Using the lazy reduction strategy sketched above, we always get a reduction of the following shape:

$$\begin{aligned}
 & p(\text{true}, \text{false}) \wedge (\Phi' \wedge \neg p(\text{true}, \text{false})) \\
 \rightarrow^{n+1} & p(\text{true}, \text{false}) \wedge (q(\Phi_1, \Phi_2) \wedge p(\neg \text{true}, \neg \text{false})) \\
 \rightarrow & p(\text{true}, \text{false}) \wedge p(q(\Phi_1, \Phi_2) \wedge \neg \text{true}, q(\Phi_1, \Phi_2) \wedge \neg \text{false}) \\
 \rightarrow & p(\text{true} \wedge (q(\Phi_1, \Phi_2) \wedge \neg \text{true}), \text{false} \wedge (q(\Phi_1, \Phi_2) \wedge \neg \text{false})) \\
 \rightarrow^k & p(\text{false}, \text{false}) \\
 \rightarrow & \text{false}
 \end{aligned}$$

where n is the number of steps applied on Φ' until a head normal form $q(\Phi_1, \Phi_2)$ is reached. This shape is completely forced by the lazy strategy; within the $n+1$ and k steps some non-determinism is present, but always $k \leq 6$. Note that reductions inside Φ_1 and Φ_2 are never permitted. By Lemma 3 we have $n \leq 2m$, so the length of the reduction is linear in m . Note that we only considered unshared rewriting. In shared rewriting however essentially the same lazy reduction is forced.

Conversely, it can be proved that for $(\dots((p_1 \text{ xor } p_2) \text{ xor } p_3) \dots) \text{ xor } p_n$ the *apply*-algorithm determines the ROBDD in time quadratic in n , while the lazy strategy admits reductions of length exponential in n . The proof is similar to that of Example 3 □

The lazy reduction appears to be similar to the *up-one* algorithm in [1]. There it is shown that for certain benchmarks *up-one* is relatively efficient, but there additional rewrite rules are used, e.g. $x \text{ xor } x \rightarrow \text{false}$. We have proved that it can also be an improvement without adding more rules. On the other hand, we gave an example on which the traditional *apply*-algorithm turned out to be better.

4 Conclusion

The TRS approach is promising, as it concisely and flexibly describes the BDD data structure and operations. Extensions to the data structure, like complemented edges, DDDs, BEDs and EQ-BDDs can be obtained basically by extending the signature. Various known algorithms are obtained as different reduction strategies. In this way the relative complexity of various proposals can be analyzed.

Acknowledgment. We want to thank Vincent van Oostrom for his contribution to the theory of sharing and for many fruitful discussions.

References

1. ANDERSEN, H. R., AND HULGAARD, H. Boolean expression diagrams. In *Twelfth Annual IEEE Symposium on Logic in Computer Science* (Warsaw, Poland, 1997), IEEE Computer Society, pp. 88–98.
2. BRYANT, R. E. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers* C-35, 8 (1986), 677–691.
3. BRYANT, R. E. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys* 24, 3 (1992), 293–318.
4. BURCH, J., CLARKE, E., LONG, D., McMILLAN, K., AND DILL, D. Symbolic model checking for sequential circuit verification. *IEEE Trans. Computer Aided Design* 13, 4 (1994), 401–424.
5. CLARKE, E., EMERSON, E., AND SISTLA, A. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8, 2 (1986), 244–263.
6. GROOTE, J., AND VAN DE POL, J. Equational binary decision diagrams. Tech. rep. SEN-R0006, CWI, Amsterdam, 2000. Available via <http://www.cwi.nl/~vdpol/papers/eqbdds.ps.Z>.
7. KLOP, J. W. Term rewriting systems. In *Handbook of Logic in Computer Science*, D. G. S. Abramski and T. Maibaum, Eds., vol. 2. Oxford University Press, 1992.
8. MEINEL, C., AND THEOBALD, T. *Algorithms and Data Structures in VLSI Design: OBDD — Foundations and Applications*. Springer, 1998.
9. MØLLER, J., LICHTENBERG, J., ANDERSEN, H. R., AND HULGAARD, H. Difference decision diagrams. In *Computer Science Logic* (Denmark, Sept. 1999).
10. PLUMP, D. Term graph rewriting. In *Handbook of Graph Grammars and Computing by Graph Transformation, volume 2: Applications, Languages* (1999), H.-J. K. H. Ehrig, G. Engels and G. Rozenberg, Eds., World Scientific, pp. 3–61.
11. VAN DE POL, J. C., AND ZANTEMA, H. Binary decision diagrams by shared rewriting. Tech. Rep. UU-CS-2000-06, Utrecht University, 2000. Also published as CWI report SEN-R0001, Amsterdam. Available via <http://www.cs.uu.nl/docs/research/publication/TechRep.html>.

Verifying Single and Multi-mutator Garbage Collectors with Owicki-Gries in Isabelle/HOL

Leonor Prensa Nieto^{*} and Javier Esparza

Technische Universität München
Institut für Informatik, 80290 München, Germany
{prensani,esparza}@in.tum.de

Abstract. Using a formalization of the Owicki-Gries method in the theorem prover Isabelle/HOL, we obtain mechanized correctness proofs for two incremental garbage collection algorithms, the second one parametric in the number of mutators. The Owicki-Gries method allows to reason directly on the program code; it also splits the proof into many small goals, most of which are very simple, and can thus be proved automatically. Thanks to Isabelle's facilities in dealing with syntax, the formalization can be done in a natural way.

1 Introduction

The Owicki-Gries proof system [11] is probably the simplest and most elegant extension of Hoare-logic to parallel programs with shared-variable concurrency. Like Hoare-logic, it is a syntax oriented method, i.e., the proof is carried out on the program's text. Moreover, it provides a methodology for breaking down correctness proofs into simpler pieces: once the sequential components of the program have been annotated with suitable assertions, the proof reduces to showing that the annotation of each component is valid in Hoare sense, and that each assertion of an annotation is invariant under the execution of the actions of the other components (so-called interference-freeness). Finally, the annotated program helps humans to understand why the algorithm works, and to gain confidence in the proof.

One problem of the method is that the number of interference-freeness tests is $O(k^n)$, where n is the number of sequential components, and k is the maximal number of lines of a component. This makes a complete pencil and paper proof very tedious, even for small examples. For this reason, many of the interference-freeness proofs, which tend to be very simple, are usually omitted. This, however, increases the possibility of a mistake. One way out of this situation is to apply a theorem prover which automatically proves the easy cases, ensures that no mistakes are made, and guarantees that the proof is complete.

In [10], the Owicki-Gries method was formalized in the theorem prover Isabelle/HOL. In this paper we show that the method and its mechanization can be successfully applied to larger examples than those considered in [10]. We study

^{*} Supported by the DFG PhD program "Logic in Computer Science".

two garbage collection algorithms. We first verify (a slightly modified version of) Ben-Ari's classical algorithm [2]. A pencil and paper proof using the Owicki-Gries method plus ad-hoc reasoning was presented in [14]. Our proof follows [14], but it manages to formulate the extra reasoning within the Owicki-Gries method. Ben-Ari's algorithm has also been mechanically proved using the Boyer-Moore prover [13] and PVS [6], but none of these proofs uses Owicki-Gries. This makes the algorithm an excellent example for comparing the Owicki-Gries method with others, and for comparing Isabelle/HOL with other theorem provers.

In the last section of the paper we verify a parametric garbage collector in which an arbitrary number of mutators work in parallel. The algorithm was proved by hand in [8] with the help of variant functions. To our knowledge this is the first mechanized proof. Notice that correctness must be shown for an infinite family of algorithms, which introduces an additional difficulty.

The paper is structured as follows: in Section §2 we briefly present our language, the Owicki-Gries method and some basic information about Isabelle/HOL. The basics of garbage collection algorithms are described in Section §3. Section §4 presents the proof of Ben-Ari's algorithm in detail. Section §5 presents the proof of the parametric algorithm. Section §6 contains conclusions.

For space reasons we only sketch the proof of the parametric algorithm, and slightly simplify the annotations of the programs. Complete annotations and proof scripts can be obtained from <http://www.in.tum.de/~prensani/>.

2 The Owicki-Gries Method and Isabelle/HOL

The Owicki-Gries proof system is an extension of Hoare-logic to parallel programs. Two new statements deal with parallel processing. The **COBEGIN**-**COEND** statement encloses processes that are executed in parallel; the **AWAIT** statement provides synchronization. We consider the evaluation of any expression or execution of any assignment as an *atomic action*, i.e., an indivisible operation that cannot be interrupted. If several instructions are to be executed atomically, they form an *atomic region*. Syntactically, these are enclosed in angled brackets $<$ and $>$.

Proofs for parallel programs are given in the form of *proof outlines*, i.e., the program is annotated at every control point where interference may occur. Given two proof outlines S and T , we say that they are *interference free* if, for every atomic action s in S with precondition $pre(s)$, and every assertion P in T , the formula $\{P \wedge pre(s)\} s \{P\}$ holds, and conversely. Thus, the execution of any atomic action cannot affect the truth of the assertions in the parallel programs. The inference rule for the verification of the **COBEGIN**-**COEND** statement is:

$$\frac{\{p_i\}S_i\{q_i\} \text{ for } i \in \{1, \dots, n\} \text{ are correct and interference free}}{\{\bigwedge_{i=1}^n p_i\} \text{ COBEGIN } S_1 \parallel \dots \parallel S_n \text{ COEND } \{\bigwedge_{i=1}^n q_i\}}$$

An important aspect of the Owicki-Gries method is the use of auxiliary variables. They augment the program with additional information for proof purposes. An auxiliary variable a is only allowed to appear in assignments of the form $a := t$, and so it is superfluous for the real computation.

Isabelle [1,12] is a generic interactive theorem prover and Isabelle/HOL is its instantiation for higher-order logic. For a tutorial introduction see [9]. We do not assume that the reader is already familiar with HOL and summarize the relevant notation: The i th component of the list xs is written $xs!i$, and $xs[i:=x]$ denotes xs with the i th component replaced by x . Set comprehension syntax is $\{e. P\}$. To distinguish variables from constants, we show the latter in **sans-serif**. We will use the syntax for while-programs as it is formalized in Isabelle/HOL: Assertions are surrounded by “{.” and “.}”. The syntax for assignments is $x ::= t$. Sequential composition is represented by a double semi-colon (;) or by a double comma (,) when it occurs inside atomic regions.

3 Garbage Collection

Garbage collection is the automatic reclamation of memory space. User processes, called *mutators*, might produce garbage while performing their computations. The *collector*’s task is to identify this garbage and to recycle it for future use by appending it to the *free list*. *Incremental* (also called on-the-fly) garbage collection systems, are those where the garbage collection work is randomly interleaved with the execution of instructions in the running programs.

The **memory** is modelled as a finite directed graph with a fixed number of nodes, where each node has a fixed set of outgoing edges. A pre-determined subset of nodes, called the **Roots**, is always accessible to the running program. A node is called *reachable* or *accessible* if a directed path exists along the edges from at least one root to that node, otherwise, it is called *garbage*. For marking purposes, each node is associated a color, which can be black or white. The memory structure can only be modified by one of the following three operations: redirect an edge from a reachable node towards a reachable node, append a garbage node to the free list, or change the color of a node.

The **mutators** abstractly represent the changes that user programs produce on the memory structure. It is assumed that they only work on nodes that are reachable, having the ability to redirect an edge to some new target. To make garbage collection safe, the mutators cooperate with the collector by assuming the overhead of blackening the new target. Thus, a mutator repeatedly redirects some edge R to some reachable node T , and then colors the node T black.

It is customary to describe the **collector**’s task in this way: identify the nodes that are garbage, i.e., no longer reachable, and append them to the free list, so that their space can be reused by the running program. However, at an abstract level it suffices to assume that the collector makes garbage nodes accessible again: since the mutator has the ability to redirect arbitrary accessible edges, it may reuse these nodes. In the sequel adding a node to the free list will just mean making it accessible.

The collector repeatedly executes two phases, traditionally called “marking phase” and “sweep” or “appending phase”. In the marking phase the collector (1) colors the roots black; (2) visits each edge, and if the source is black it colors the target black; (3) counts the black nodes; (4) if not all reachable nodes are

black, goes to step (2). In the appending phase, the collector (5) visits each node, appending white nodes to the free list, and coloring black nodes white. The safety property we prove says that *no reachable node is garbage collected*. In other words, if during the appending operation a node is white, then it is garbage. Clearly, this property holds if step 4 is correct. But how do we determine that all reachable nodes are black? In the case of one mutator, Ben Ari's solution is to keep the result of the last count, and compare it with the result of the current count. If they coincide, then all reachable nodes are black. For n mutators, we compare the results of the last $n+1$ counts. So the algorithms for one and several mutators differ only in step 4.

4 The Single Mutator Case

We verify (a slightly modified version of) Ben-Ari's algorithm. We follow the ideas of [14], but formulate the proof completely within the Owicki-Gries system.

The Memory. The memory is formalized using two lists of fixed size. In the first list, called **M**, memory nodes are indexed by natural numbers that range from 0 to the length of **M**; the color of node **i** can be consulted by accessing **M[i]**. The second list, called **E**, models the edges; each edge is a pair of natural numbers corresponding to the source and the target nodes. **Roots** is an arbitrary set of nodes. **Reach** is the set of nodes reachable from **Roots** (including **Roots** itself). **Blacks** is the set of nodes that are **Black**. Finally, **BtoW** are the edges that point from a **Black** node to a **White** node.

The separate treatment of colors and edges in our data structure is an abstraction that considerably simplifies proofs relating to the changes in the graph. If an edge is redirected, **M** remains invariant, while coloring does not modify **E**.

The Mutator. The auxiliary variable **z** is false if the mutator has already redirected an edge but has not yet colored the new target. Some obvious conditions

```
{.T ∈ ReachE ∧ z.} WHILE True
INV {.T ∈ ReachE ∧ z.} DO <
E ::= E[R := (fst(E!R), T)],, z ::= ¬z >;
{.T ∈ ReachE ∧ ¬z.} < M ::= M[T := Black],,
z ::= ¬z > OD {.False.}
```

Fig. 1. The mutator

required of the selected edge **R** and node **T**, namely, $R < \text{length } E$ and $T < \text{length } M$ always hold and are omitted in the annotated program text. The verification requires to prove one lemma: an accessible node cannot be rendered inaccessible by redirecting an edge to it.

The Collector. The collector first blackens the roots and then executes a loop. The body of the loop consists of first traversing **M** coloring all reachable nodes black, and then counting the number of black nodes. The loop terminates if the results of the current count and the previous one coincide. After termination of the loop, the collector traverses **M** once more, this time making all white nodes reachable and all black nodes white. We divide the algorithm into

modules, which are pieces of code together with their pre- and postconditions. The `Blackening_Roots` module is straightforward; the codes and annotations of the rest are explained separately. Obvious intermediate assertions are omitted.

```

{. True. }
WHILE True INV { . True. }
DO Blackening_Roots;
{ . Roots  $\subseteq$  Blacks M. }
OBC ::= {}; BC ::= Roots; Ma ::= L;
WHILE OBC  $\neq$  BC
  INV { . Roots  $\subseteq$  Blacks M
         $\wedge$  OBC  $\subseteq$  Blacks Ma  $\subseteq$  BC  $\subseteq$  Blacks M
         $\wedge$  (Safe(M, E)  $\vee$  OBC  $\subseteq$  Blacks Ma ) . }
  DO OBC ::= BC; Propagating_Black;
    Ma ::= M; BC ::= {}; Counting
  OD;
{ . Safe(M, E) . }
Appending
OD { . False. }

```

Fig. 2. The collector

The variables `BC` (Black Count) and `OBC` (Old Black Count) are used to determine if the set of black nodes has grown during the last `Propagating_Black` phase. Following [14], `OBC` is initialized to $\{\}$, and `BC` to the set `Roots`¹. A single auxiliary variable `Ma` is used for “recording” the value of `M` after the execution of `Propagating_Black`. The constant `L` is used to give `Ma` a suitable first value, defined as a list of nodes where only the `Roots` are black.

```

{ . Roots  $\subseteq$  Blacks M
   $\wedge$  OBC  $\subseteq$  BC  $\subseteq$  Blacks M. }
I := 0;
WHILE I < length E
  INV { . Roots  $\subseteq$  Blacks M
         $\wedge$  OBC  $\subseteq$  BC  $\subseteq$  Blacks M
         $\wedge$  PB(M, E, OBC, I, z)
         $\wedge$  I  $\leq$  length E . }
  DO IF M! (fst(E!I)) = Black THEN
    M ::= M[snd(E!I) := Black] FI;
    I := I + 1
  OD
{ . Roots  $\subseteq$  Blacks M
   $\wedge$  OBC  $\subseteq$  BC  $\subseteq$  Blacks M
   $\wedge$  (OBC  $\subseteq$  Blacks M  $\vee$  Safe(M, E)) . }

```

Fig. 3. Module `Propagating_Black`

guarantees that, if an execution of the body does not establish the safety property, then `OBC` is a proper lower bound, which means that some white node was colored black during the execution of `Propagating_Black`. The `Propagating_Black` and `Counting` modules have very clear tasks: `Propagating_Black` establishes the third conjunct, while `Counting` establishes the second.

`Safe(M, E)` states that all reachable nodes are black, i.e., $\text{Reach } E \subseteq \text{Blacks } M$. Since we have `Safe(M, E)` before `Appending`, all white nodes are garbage right before the appending module starts. This is almost the safety property we wish to prove, since, as we shall show later when describing the `Appending` module, if a white node is garbage before `Appending`, then it remains so until `Appending` makes it reachable.

The key parts of the invariant are the second and third conjuncts. The second conjunct guarantees that after any execution of the body the cardinalities of `OBC` and `BC` are a lower and upper bound, respectively, of the number of black nodes after `Propagating_Black`. (It is clear that `OBC` is a lower bound, because black nodes stay black until the beginning of the appending phase. That `BC` is an upper bound is the difficult part, since the mutator can blacken nodes while the collector executes `Counting`.) The third conjunct

¹ `OBC` and `BC` are here sets of black nodes whereas in the original algorithm they represent their cardinalities. We found the set approach easier to formalize but it simplifies neither the algorithm nor the proofs.

Propagation of the Coloring. During this phase, the collector visits the edges in a given order, coloring the target whenever the source was **Black**. This phase establishes the third conjunct of the invariant.

The invariant of this module is tricky. The predicate PB is an adaptation of the one proposed in [14]. $PB(M, E, OBC, I, z)$ denotes the predicate

$$OBC \subseteq \text{Blacks } M \vee (\forall i < I. \neg \text{BtoW}(E!i, M) \vee (\neg z \wedge i = R \wedge \text{snd}(E!R) = T \wedge (\exists r < \text{length } E. I \leq r \wedge \text{BtoW}(E!r, M))))$$

and it is the crux of the proof. Intuitively, its invariance is proved as follows. If the collector or the mutator blacken some white node, then after execution of the body $OBC \subseteq \text{Blacks } M$ holds. If all the edges visited by the collector point to a **Black** node, then $\forall i < I. \neg \text{BtoW}(E!i, M)$ holds. If some visited edge points to a white node (because the mutator has redirected it), then (and this is Ben-Ari's main observation) there is another **BtoW** edge among those that have not yet

been visited: since the white node T is reachable, there is a path to T from some root, and since all roots are **Black**, some edge along this path must be a **BtoW** edge. Observe that upon termination of the loop this last clause cannot hold since $I = \text{length } E$. To obtain the postcondition $OBC \subseteq \text{Blacks } M \vee \text{Safe}(M, E)$ we need to prove a lemma: if all **Roots** are **Black** and no edge points from a **Black** node to a **White** node, then all reachable nodes are **Black**.

```

{.Roots ⊆ Blacks M
  ∧ OBC ⊆ Blacks Ma ⊆ Blacks M
  ∧ BC = {}
  ∧ (OBC ⊆ Blacks Ma ∨ Safe(M, E)).}
I := 0;;
WHILE I < length M
  INV { .Roots ⊆ Blacks M
        ∧ OBC ⊆ Blacks Ma ⊆ Blacks M
        ∧ BC ⊆ Blacks M
        ∧ (OBC ⊆ Blacks Ma ∨ Safe(M, E))
        ∧ {i. i < I ∧ Ma!i = Black} ⊆ BC
        ∧ I ≤ length M. }
  DO IF M!I = Black THEN BC := (insert I BC) FI;;
  I := I + 1
OD
{.Roots ⊆ Blacks M
  ∧ OBC ⊆ Blacks Ma ⊆ BC ⊆ Blacks M
  ∧ (OBC ⊆ Blacks Ma ∨ Safe(M, E)).}

```

Fig. 4. Module Counting

Counting Black Nodes. This phase finally re-establishes the invariant of the collector's outermost loop. The computed set BC must contain all nodes which were black upon termination of *Propagating-Black*, or, since Ma records precisely this

```

{. Safe(M, E). }
I := 0;;
WHILE I < length M
  INV { . Safe.I(M, E, I) ∧ I ≤ length M. }
  DO IF M!I = Black THEN M := M[I := White]
      ELSE { . I ∉ ReachE ∧ Safe.I(M, E, I). }
      E := AppendToFree(I, E)
      FI;; I := I + 1
OD { . True. }

```

Fig. 5. Module AppendToFree

set, the *Counting* phase must ensure that $\text{Blacks } Ma \subseteq BC$ holds. Since the mutator can only blacken nodes, this task is now trivial. With all reachable nodes marked we can proceed to the appending phase where all unmarked nodes are appended to the free list.

Appending to the Free List. Here we follow our predecessors: Appending a garbage node I to the free list (i.e. making I reachable) is modelled by an abstract function `AppendtoFree` satisfying suitable axioms. In the annotated code, `Safe_I(M,E,I)` states that all white nodes with index I or larger are garbage. The precondition of the assignment to E guarantees that only garbage nodes are collected (the conjunct `Safe_I(M,E,I)` is needed here to maintain the invariant throughout the loop).

5 The Multi-mutator Case

If we allow the interaction with several mutators, new difficulties come into play. We consider a solution, first presented in [8], in which the collector proceeds to the appending phase only after $n+1$ consecutive executions of the `Propagating_Black` phase during which the set of black nodes did not increase. Observe that in the case of one mutator this collector checks *twice* whether `OB=BC`, and not only once, as the collector of Section §4. In [8] it is shown that n consecutive executions suffice, but we do not consider this version in the paper. The program consists of a fixed, finite and nonempty set of mutator processes and one collector process. When the number of programs is a parameter, the list

of programs to be executed in parallel can be expressed using the function `map` and the construct `[i..j]`, which represents the list of natural numbers from i to j (the syntax `[i..j()]` corresponds to `[i..j-1]`).

The syntax and the tactic for the generation of the verification conditions presented in [10] have been extended to deal with this kind of program schemas. They are preceded by the word `SCHEME`.

```
SCHEME map [λj.
{.Z (Muts!j).}
WHILE True
  INV { .Z (Muts!j) .}
DO < IF T (Muts!j) ∈ ReachE THEN
  E := E[R (Muts!j) := (fst(E!(R (Muts!j)))),
  T (Muts!j))] FI ,,
  Muts := Muts[j := (Muts!j) (IZ := False)] > ;;
{.¬Z (Muts!j).}
< M := M[T (Muts!j) := Black] ,,
  Muts := Muts[j := (Muts!j) (IZ := True)] >
OD { .False. } )
[0..n()
```

Fig. 6. The mutators

The Mutators. A mutator can only redirect an edge when its target is a reachable node, and redirecting may make its old target inaccessible. If several mutators are active, then one of them may select a reachable node T as new target, but another one may render T inaccessible before the edge has been redirected to T . To solve this problem, selecting the new

target and redirecting the edge is modelled as a single atomic action.

Each mutator m selects an edge R_m and a target node T_m . As in the previous section each mutator owns an auxiliary variable Z_m that indicates when the mutator is pending before the blackening of a node. These three objects are put together in a record. Isabelle's syntax for accessing the field Z of a record variable `Mut` is `Z Mut`. Record update is written `Mut (IZ := True)`, meaning that the field Z of the record `Mut` is updated to the value `True`. The variable `Muts` is a

list of length n (the number of mutators) whose components are records of type `mut`. For example, to access the selected edge of mutator j we write $R (Muts!j)$.

The Collector. In the case of one mutator, if an execution of the body does not establish the safety property, then some white node was colored black during the execution of `Propagating_Black`. When several mutators are present, there may be other reasons. To describe them we need a new value `Queue(Muts,M)` which represents the number of mutators that are *queueing* to blacken a white node.

```

{. True.}
WHILE True INV { . True.}
DO Blackening_Roots;;
  OBC::={}; BC::=Roots;; l::=0;;
  WHILE l < n+1
  INV { . Roots ⊆ BlacksM
    ∧ OBC ⊆ BC ⊆ BlacksM
    ∧ (Safe(M,E)
      ∨ ( (BC ⊆ BlacksM
        ∨ l ≤ Queue(Muts,M)
        ∧ l < n+1)) ) }
  DO OBC ::= BC;; Propagating_Black;;
    < Ma ::= M, ,Qa ::= Queue(Muts,M) >;
    BC ::= {}; Counting;;
    { . Roots ⊆ BlacksM
      ∧ OBC ⊆ BlacksMa ⊆ BC ⊆ BlacksM
      ∧ (Safe(M,E)
        ∨ OBC ⊆ BlacksMa
        ∨ (l < Qa ∧ OBC ⊆ BlacksM)
        ∨ (l < Qa ∧ Qa ≤ Queue(Muts,M)))
      ∧ Qa < n+1. }
    IF OBC = BC THEN l ::= l+1 ELSE l ::= 0 FI
  OD;;
  { . Safe(M,E). }
  Appending
OD { . False. }

```

Fig. 7. The collector

The auxiliary variable `Qa` will “record” this value upon termination of the `Propagating_Black` phase. The invariant of the one mutator case must be compared with the precondition of the `IF-THEN-ELSE` instruction, because both correspond to the assertion established by the `Counting` phase. The assertion $\text{Safe}(M,E) \vee OBC \subset \text{Blacks } Ma$ has been weakened with new disjuncts, corresponding to the new situations which can prevent $\text{Safe}(M,E)$ from holding. The first new disjunct corresponds to the case in which at least one mutator joins `Queue(Muts,M)` and then colors its new target during `Counting`. The second conjunct corresponds to the case in which no

mutator performs any coloring. Intuitively, after $n+1$ non-blackening `Propagating_Black` iterations, the property $\text{Safe}(M,E)$ must hold, since the number of queueing mutators cannot exceed n .

The codes of the modules are the same as in §4 up to annotations in the `Propagating_Black` and `Counting` phases, which have to be adapted to the new invariant.

We just show the invariant of the `Propagating_Black` phase:

```

{ . Roots ⊆ Blacks M ∧ OBC ⊆ BC ⊆ Blacks M ∧ l ≤ length E.
  ∧ ( Safe(M,E) ∨ OBC ⊆ Blacks M ∨ l < Queue(Muts,M)
    ∨ (∀ i < l. ¬ BtoW(E!i,M) ∧ l ≤ Queue(Muts,M)) ) }

```

Any coloring establishes $OBC \subset \text{Blacks } M$. (Observe that only coloring can make the queue shorter.) If no coloring occurs then either all the visited edges point to a black node, or some mutator has redirected an edge to a white source but has not yet colored the target, which amounts to saying that the queue grows ($l < \text{Queue}(Muts,M)$).

6 Conclusions and Related Work

The Owicki-Gries method splits the proof into a large number of simple interference-freeness subproofs. These are very tedious to prove by hand, and so avoided by humans, who prefer to split a proof into a few difficult cases. In order to investigate if the use of a theorem prover can palliate this problem, we have provided mechanically checked Owicki-Gries proofs for two garbage collection algorithms. The result is: 320 out of 340 interference-freeness proofs in the final annotations were automatically carried out by Isabelle/HOL. For the remaining 20 interference-freeness proofs only three lemmas had to be supplied. The proofs of these lemmas, however, were very interactive.

We do not know of any complete Owicki-Gries proof for any of the two algorithms. In his proof of Ben-Ari's algorithm [14], van de Snepscheut mixes the Owicki-Gries method with ad-hoc reasoning; in particular, he does not provide an invariant for the outermost loop, implicitly claiming that doing so will be complicated. However, the invariant turns out to be simple (3 clauses), and has a clear intuitive interpretation. In [8], Jonker argues that "A proof [of the n -mutators algorithm] according to the Owicki-Gries theory would require the introduction of a satisfactory number of ghost variables In an earlier version of this paper the invariant we constructed was rather unwieldy and the proof of invariance almost unreadable." However, our proof only uses two auxiliary variables (Ma and Qa), plus a trivial auxiliary variable for each mutator. Extending our proof to the more elaborated n -mutator algorithms of [8] should be possible with reasonable effort.

We know of two other mechanized proofs of Ben-Ari's algorithm, carried out using the Boyer-Moore theorem prover [13] and PVS [6,7]. The main advantage of our approach is probably the closeness to the original program text, which simplifies the interaction with the prover: Annotated programs are rather readable by humans, and they are also directly accepted as input by Isabelle. In other approaches the program must be first translated into a different language (e.g. LISP in [13]).

Another aspect of our formalization is that we only had to prove 8 lemmas (3 of them trivial) about graph functions, whereas 100 lemmas were required in [13], and about 55 in [6,7]. The reason for this is that many trivial lemmas about sets or lists could be automatically proved using Isabelle's built-in tactics (rewriting, classical reasoning, decision procedures for Presburger arithmetic, etc) and Isabelle's standard libraries. The proof effort, however, took two months for the one-mutator algorithm (similar to our predecessors) and another two months for the n -mutator case. Most of the time was consumed in finding and improving the invariants.

A disadvantage of the Owicki-Gries method (in its classical version) is that it can only be applied to safety properties, while in [8,13,14] the liveness property "every garbage node is eventually collected" is also proved to hold.

None of our two algorithms has been proved correct using fully automatic methods. In [3] there is a proof of Ben Ari's algorithm for 1 mutator and 4 memory cells. In [4], a predecessor of Ben Ari's algorithm is proved correct using

automatic tools for generating and proving invariants. The key invariants, however, require intelligent input from the user. The paper suggests using predicate abstraction for checking or strengthening invariants in a larger verification effort involving interactive theorem provers, which is a promising idea.

Our overall conclusion is that the application of a theorem prover greatly enhances the applicability of the Owicki-Gries method. The closeness to the original program is preserved, and the large number of routine proofs is considerably automatized.

References

1. Isabelle home page. www.cl.cam.ac.uk/Research/HVG/isabelle.html.
2. M. Ben-Ari. Algorithms for on-the-fly garbage collection. *ACM Toplas*, 6:333–344, 1984.
3. G. Bruns. *Distributed Systems Analysis with CCS* Prentice-Hall, 1997.
4. S. Das, D. L. Dill and S. Park. Experience with predicate abstraction. In CAV '99, LNCS 1633, 160–171, 1999.
5. E. W. Dijkstra, L. Lamport, A. J. Martin, C. S. Scholten and E. F. M. Steffens. On-the-fly garbage collection: An exercise in cooperation. *Communications of the ACM*, 21(11):966–975, 1978.
6. K. Havelund. Mechanical verification of a garbage collector. FMPPTA'99. Available at <http://ic-www.arc.nasa.gov/ic/projects/amphion/people/havelund/>.
7. K. Havelund and N. Shankar. A mechanized refinement proof for a garbage collector. *Formal Aspects of Computing*, 3:1–28, 1997.
8. J. E. Jonker. On-the-fly garbage collection for several mutators. *Distributed Computing*, 5:187–199, 1992.
9. T. Nipkow. *Isabelle/HOL. The Tutorial*, 1998. Unpublished Manuscript. Available at www.in.tum.de/~nipkow/pubs/HOL.html.
10. T. Nipkow and L. Prensa Nieto. Owicki/Gries in Isabelle/HOL. In FASE'99, LNCS 1577, 188–203. Springer-Verlag, 1999.
11. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica*, 6:319–340, 1976.
12. L. C. Paulson. *Isabelle: A Generic Theorem Prover*, LNCS 828 Springer-Verlag, 1994.
13. D. M. Russinoff. A mechanically verified garbage collector. *Formal Aspects of Computing*, 6:359–390, 1994.
14. J. L. A. van de Snepscheut. “Algorithms for on-the-fly garbage collection” revisited. *Information Processing Letters*, 24:211–216, 1987.

Why so Many Temporal Logics Climb up the Trees?

Alexander Rabinovich and Shahar Maoz

School of Mathematical Science
Tel Aviv University, Tel Aviv, Israel 69978
`{rabino, maoz}@math.tau.ac.il`

Abstract. Many temporal logics were suggested as branching time specification formalisms during the last 20 years. These logics were compared against each other for their expressive power, model checking complexity and succinctness. Yet, unlike the case for linear time logics, no canonical temporal logic of branching time was agreed upon. We offer an explanation for the multiplicity of temporal logics over branching time and provide an objective quantified ‘yardstick’ to measure these logics.

We define an infinite hierarchy BTL_k of temporal logics and prove its strictness. We show that CTL^* has no finite base, and that almost all of its many sub-logics suggested in the literature are inside the second level of our hierarchy. We show that for every logic based on a finite set of modalities, the complexity of model checking is linear both in the size of structure and the size of formula.

1 Introduction

Various temporal logics have been proposed for reasoning about so-called “reactive” systems, computer hardware or software systems which exhibit (potentially) non-terminating and non-deterministic behavior. Such a system is typically represented by the (potentially) infinite sequences of computation states through which it may evolve, where we associate with each state the set of atomic propositions which are true in that state, along with the possible next state transitions to which it may evolve. Thus its behavior is denoted by a (potentially) infinite rooted tree, with the initial state of the system represented by the root of the tree.

Temporal Logic (TL) is a convenient framework for the specification properties of systems. This made TL a popular subject in the Computer Science community and it enjoyed extensive research during the last 20 years. In temporal logic the relevant properties of the system are described by *Atomic Propositions* that hold at some points in time and not at others. More complex properties are described by formulas built from the atoms using Boolean connectives and *Modalities* (temporal connectives): an l -place modality C transforms statements $\varphi_1, \dots, \varphi_l$ on points possibly other than the given point t_0 to a statement $C(\varphi_1, \dots, \varphi_l)$ on the point t_0 . The rule that specifies when is the statement

$C(\varphi_1, \dots, \varphi_l)$ true for the given point is called *Truth Table*. The choice of the particular modalities with their truth table determines the different temporal logics. A Temporal Logic with modalities $M_1, \dots, M_r$ is denoted by $TL(M_1, \dots, M_r)$.

The most basic modality is the one place modality **F** X saying “ X holds some time in the future”. Its truth table is usually formalized by $\varphi_{\mathbf{F}}(t_0, X) \equiv (\exists t > t_0)X(t)$. This is a formula of the Monadic Logic of Order (*MLO*). *MLO* is a fundamental formalism in Mathematical Logic. Its formulas are built using atomic propositions $X(t)$, atomic relations between elements $t_1 = t_2$, $t_1 < t_2$, Boolean connectives, first-order quantifiers $\exists t$ and $\forall t$, and second-order (set) quantifiers $\exists X$ and $\forall X$. Practically all the modalities used in the literature have their truth tables defined in *MLO* and as a result every formula of a temporal logic translates directly into an equivalent formula of *MLO*. Therefore, the different temporal logics may be considered a convenient way to use fragments of *MLO*. *MLO* can also serve as a yardstick by which to check the strength of the temporal logic chosen: a temporal logic is *expressively complete* for a fragment L of *MLO* if every formula of L with single variable t_0 is equivalent to a temporal formula.

Actually the notion of expressive completeness refers to a temporal logic and to a model (or a class of models) since the question if two formulas are equivalent depends on the domain over which they are evaluated. Any ordered set with monadic predicates is a model for *TL* and *MLO*, but the main, *canonical*, linear time intended models are the non-negative integers $\langle \mathbb{N}, < \rangle$ for discrete time and the non-negative reals $\langle \mathbb{R}^+, < \rangle$ for continuous time.

A major result concerning *TL* is Kamp’s theorem [13,9] which states that the pair of modalities *X until Y* and *X since Y* is expressively complete for the first-order fragment of *MLO* over the above two linear time canonical models.

There is an important distinction between the future and the past. It is usually assumed that any particular point of time has one linear past, but perhaps various futures. This might be the reason that most of the temporal formalisms studied in computer science use only future time constructs. Fortunately, Kamp’s theorem also implies that the *TL* with one modality **U** (until) has the same expressive power (over the canonical linear discrete model) as the future fragment of the first-order monadic logic. In this paper we will deal only with future fragments of *MLO* and future time temporal logics.

Milner and Park [16,18] pointed out that for the specification of concurrent systems we need a model finer than just the set of possible (linear) runs; this led to the computational tree model.

Of course, $TL(\mathbf{U})$ is interpreted not only over linear orders but over arbitrary partial orders, in particular over the trees. However, the expressive power of $TL(\mathbf{U})$ over the trees is very limited. For instance, a very basic property “for all paths that start at t_0 eventually p holds” is not expressible in $TL(\mathbf{U})$.

In order to reflect branching properties of computations many temporal logics were suggested starting from [14,1]. The basic modalities of these logics (which are often called branching time logics) are either of the form **E** “there exists a linear run” followed by a formula in $TL(\mathbf{U})$ or of the form **A** “for every linear run” followed by a formula in $TL(\mathbf{U})$. **E** φ (respectively **A** φ) holds at a moment t_0 if

for some path π (respectively, for every path π) starting at t_0 the $TL(\mathbf{U})$ formula φ holds along π . For example, one commonly used branching time logic is CTL [1]. It is based on two binary modalities $\mathbf{EU}$ and $\mathbf{AU}$; $\mathbf{AU}(X, Y)$ (respectively $\mathbf{EU}(X, Y)$) holds at a current moment t_0 if “for all (respectively, for some) runs from the current moment, X until Y holds”. In contrast to expressive completeness of $TL(\mathbf{U})$ over the canonical linear models, there is no natural predicate logic which corresponds to $TL(\mathbf{EU}, \mathbf{AU})$ (i.e., to CTL) over the trees. Moreover, it turns out that CTL cannot express many natural fairness properties.

The logic CTL^* suggested in [7] has the same expressive power as the temporal logic with infinite set of modalities $\{\mathbf{E}\varphi : \varphi \text{ is a formula of } TL(\mathbf{U})\}$. Many temporal logics were suggested as branching time specification formalisms (see [8,4]) by imposing some syntactical restrictions on CTL^* formulas. The lack of a yardstick was emphasized by Emerson in [4,5]:

“Hundreds perhaps thousands of papers developing the theory and application of temporal logic to reasoning about reactive systems were written. Dozens if not hundreds of systems of temporal logic have been investigated, both from the standpoint of basic theory and from the standpoint of applicability to practical problems . . . there is now a widespread consensus that some type of temporal logic constitutes a superior way to specify and reason about reactive systems. There is no universal agreement on just which logics are best . . . While less is known about comparisons of Branching time logics against external “yardsticks”, a great deal is known about comparisons of $BTLs$ against each other. This contrasts with the reversed situation for Linear-Time Logics.”

Our results offer an explanation for the multiplicity of temporal logics over branching time and suggest some yardsticks by which to measure these logics.

Two of the most important characteristics of a TL are (1) its expressive power and (2) the complexity of its model checking problem [5]. We examine two very natural fragments of MLO and prove that there is no temporal logic over finite basis which is expressively equivalent over trees to each of these fragments. On the other hand, we show that for every finite set of modalities $M_1, M_2, \dots, M_r$, the complexity of model checking for $TL(M_1, M_2, \dots, M_r)$ is linear both in the size of structure and the size of formula. We believe, therefore, that these are the reasons for so many suggestions for temporal logics over branching time models.

One popular equivalence between computation trees is that of bisimulation equivalence. This equivalence catches subtle differences between trees based on their branching structures. It is generally regarded as the finest behavioral equivalence of interest for concurrency. In [17], CTL^* was shown to be expressively equivalent to the bisimulation invariant fragment of monadic path logic [10]. The syntax of monadic path logic is the same as that of monadic second-order logic. The bound set (monadic) variables ranges over all the paths and semantically this logic is very closely related to the first-order logic [17]. Thus at least CTL^* represents some objectively quantified expressive power.

We describe a sequence BTL_k ($k \in Nat$) of temporal logics. All these logics are sub-logics of CTL^* and their union has the same expressive power as CTL^* .

Roughly speaking the modalities of BTL_k correspond to formulas with quantifier depth at most k . However, for every m and k there is a BTL_k formula which is equivalent to no MLO formula with quantifier depth $\leq m$. We show that BTL_{k+1} is strictly more expressive than BTL_k . Consequently, we obtain that there is no finite basis for CTL^* and hence there is no finite basis for the bisimulation invariant fragment of monadic path logic. Our proof also demonstrates that in contrast to the linear time case, there is no finite base temporal logic with the same expressive power (over the trees) as the bisimulation invariant fragment of first-order logic.

We examine the expressive power of commonly used branching time temporal logics. It turns out that almost all of these logics are inside the second level of our hierarchy. The modalities for these logics were suggested by desire to formalize some pragmatic properties which often occur in specifications of hardware and software systems. It is interesting to observe that most of these properties can be formalized by formulas with quantifier depth at most two.

The problem whether a formula φ is satisfiable in the computational tree which corresponds to a finite state (Kripke) structure is known as the model checking problem. We prove that the model checking problem has $O(|K| \times |\varphi|)$ time complexity for every temporal logic based on a finite set of modalities definable in CTL^* .

The paper is organized as follows. In the next section we review basic definitions about monadic logic of order and temporal logics. In Sect. 3 we introduce the sequence BTL_k of temporal logics. Our main technical result is that $\{BTL_k\}_{k=1}^\infty$ contains a strict hierarchy. In Sect. 4 we show that CTL^* has no finite basis and examine the expressive power of some commonly used branching time logics. In Sect. 5 we discuss the complexity of the model checking problem.

Because of space limitation the proofs are omitted. Detailed proofs can be found in [15].

2 Preliminaries

In this section we review basic definitions about computation trees, the monadic logic of order, temporal logics and modalities.

2.1 Computation Trees and Paths

A **tree** $T = (|T|, \leq)$ consists of a partially ordered set of nodes $|T|$ in which the predecessors of any given element $a \in |T|$ constitute a finite total order with a common minimal element ϵ_T , referred to as the root of the tree. A **computation tree** is a structure $(|T|, \leq, P_1, P_2, \dots)$, where $(|T|, \leq)$ is a tree, and $P_1, P_2, \dots$ are subsets of $|T|$. We say that a node $a \in |T|$ is labeled by P_i if $a \in P_i$. We often write $\mathbf{P}$ for the sequence $P_1, P_2, \dots$; the length of $\mathbf{P}$ is denoted by $length(\mathbf{P})$.

A **path** π through T , starting at an element $s_1 \in |T|$, is a maximal linearly ordered sequence of successive nodes $\pi = \langle s_1, s_2, \dots \rangle$ through T , ordered by $\leq$.

Let $T = (|T|, \leq, \mathbf{P})$ be a computation tree and let $a \in |T|$. We shall use $T_{\geq a}$ to denote the subtree of T rooted at the node a . Formally, $T_{\geq a} = (|T_{\geq a}|, \leq, \mathbf{P}')$ where $|T_{\geq a}| = \{s : s \in |T| \text{ and } s \geq a\}$ and $P'_i = P_i \cap |T_{\geq a}|$ for $i \leq \text{length}(\mathbf{P})$. We use T_π for the substructure of T over the nodes in the path π .

2.2 Monadic Logic of Order

The syntax of the **second-order Monadic Logic of Order (MLO)** has in its vocabulary individual first-order variables $x_0, x_1, x_2, \dots$ (representing nodes, states or time points), set variables $X_0, X_1, X_2, \dots$ (representing sets of nodes), and set constants (monadic predicates) $P_0, P_1, P_2, \dots$. Formulas are built up from atomic formulas of the form $x_1 = x_2, x_1 < x_2, x \in X$ and $x \in P_i$, using the Propositional connectives $\wedge$ and $\neg$, and the quantifiers $\exists x$ and $\exists X$.

We shall write $\varphi(x_1, x_2, \dots, x_m; X_1, X_2, \dots, X_n)$ to indicate that the variables $x_1, x_2, \dots, x_m; X_1, X_2, \dots, X_n$ may appear free in φ .

The **quantifier depth** of a formula φ , denoted by $\text{qd}(\varphi)$, is defined as usual. We write $(T, s_1, \dots, s_m, S_1, \dots, S_n) \models \varphi(x_1, \dots, x_m, X_1, \dots, X_n)$ if the formula $\varphi(x_1, \dots, x_m, X_1, \dots, X_n)$ is satisfied in the tree T with x_i interpreted by the node s_i ($1 \leq i \leq m$) and X_j interpreted by the set of nodes S_j ($1 \leq j \leq n$).

We shall denote by **FOMLO** the subset of first-order formulas of **MLO** that do not have set quantification. We also consider Monadic Path Logic **MPL** [10]. Its syntax is the same as that of monadic second-order logic. However, the bound set (monadic) variables range over all the paths (not over arbitrary sets of nodes) and semantically it is very closely related to the first-order logic [17].

Definition 1 (Future MLO Formula). A formula $\varphi(x_0, X_1, X_2, \dots, X_l)$ of **MLO** with one free first-order variable x_0 , is a future formula, if for every computation tree T and node $a \in |T|$, and every l subsets $S_1, S_2, \dots, S_l$ of $|T|$, the following holds:

$T, a, S_1, \dots, S_l \models \varphi(x_0, X_1, \dots, X_l)$ iff $T_{\geq a}, a, S'_1, \dots, S'_l \models \varphi(x_0, X_1, \dots, X_l)$ where $S'_i = \{s : s \in S_i \text{ and } s \geq a\}$ for all $i, i = 1, 2, \dots, l$. First-order future formulas are defined similarly.

2.3 Temporal Logics and Modalities

In the following, we recall the syntax and semantics of temporal logics and how temporal modalities are defined using **MLO** truth tables, with notation adopted from [9,11].

The syntax of **Temporal Logic (TL)** has in its vocabulary a set of **predicate variables** $\{q_1, q_2, \dots\}$ and a set B of **modality names** (sometimes called “temporal connectives” or “temporal operators”) with prescribed arity $B = \{\#_1^{(l_1)}, \#_2^{(l_2)}, \dots\}$ (we usually omit the arity notation). The set of modality names B might be infinite. The syntax of $TL(B)$ is given by the following grammar:

$$\varphi ::= q_i \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi_1 \mid \#_i^{l_i}(\varphi_1, \varphi_2, \dots, \varphi_{l_i})$$

Temporal formulas are interpreted over partially ordered sets, in particular over computation trees. Every modality $\#^{(l)}$ is interpreted in every tree T as an operator $\#_T^{(l)} : [P(T)]^l \rightarrow P(T)$ which assigns “the set of points where $\#^{(l)}[P_1, \dots, P_l]$ holds” to the l -tuple $\langle P_1, \dots, P_l \rangle$. Formally, the semantics of a formula $\varphi \in TL$ over a tree $T = (\mid T \mid, \leq, \mathbf{Q})$ is defined inductively as follows. For atomic formulas $T, s \models q_i$ iff $s \in Q_i$; the semantics of Boolean combinations is defined as usual, and the semantics of modalities is defined by: $T, s \models \#^{(l)}(\varphi_1, \varphi_2, \dots, \varphi_l)$ iff $s \in \#_T^{(l)}(R_{\varphi_1}, R_{\varphi_2}, \dots, R_{\varphi_l})$ where $R_{\varphi_i} = \{a : T, a \models \varphi_i\}$ for all $i, 1 \leq i \leq l$.

In this paper, we consider only temporal modalities which are defined in *MLO*: we assume that for every l place modality $\#$ there is a formula (truth table) $\bar{\#}(x_0, X_1, X_2, \dots, X_l)$ of *MLO* with one free first-order variable x_0 and l set variables, such that for every tree T and subsets $Q_i \subseteq \mid T \mid$:

$$\#_T(Q_1, Q_2, \dots, Q_l) = \{s : \mid T \mid, \leq, s, Q_1, Q_2, \dots, Q_l \models \bar{\#}[x_0, X_1, X_2, \dots, X_l]\}$$

Example 2 (Some common modalities and their truth tables).

- The one place modality **Fq** (“eventually q ”); its truth table is $\varphi(x_0, X) \equiv \exists y(y > x_0 \wedge y \in X)$.
- The one place modality **Gq** (“globally q ”); its truth table is $\varphi(x_0, X) \equiv \forall y(y > x_0) \rightarrow (y \in X)$.
- The two place modality **U**(q_1, q_2) (“ q_2 until q_1 ”); its truth table is $\varphi(x_0, X, Y) \equiv \exists y(y > x_0 \wedge y \in X \wedge \forall z(x_0 < z < y) \rightarrow z \in Y)$. In the literature, some times “non strict” definition of Until is given: the “non-strict until” **U_{ns}**(q_1, q_2) modality has truth table $\varphi(x_0, X, Y) \equiv \exists y(y \geq x_0 \wedge y \in X \wedge \forall z(x_0 \leq z < y) \rightarrow z \in Y)$. Clearly, **U_{ns}** can be defined using **U**.

The choice of the particular modalities with their truth tables determines the different temporal logics. The following are standard definitions and notation to discuss comparative expressive power between two temporal logics TL_1 and TL_2 .

Definition 3. Let C be a set of structures. TL_1 is **less or equally expressive** than TL_2 over C (notation $TL_1 \preceq_{exp}^C TL_2$), if for any formula $\varphi_1 \in TL_1$ there is a formula $\varphi_2 \in TL_2$ which is equivalent to φ_1 over C . The relations **equally expressive** (notation $\equiv_{exp}^C$) and **strictly less expressive** (notation $\prec_{exp}^C$) are defined from the relation $\preceq_{exp}^C$ as expected. When C is the class of trees, we write $\preceq_{exp}$ for $\preceq_{exp}^C$. Similarly for $\equiv_{exp}$ and $\prec_{exp}$.

Definition 4 (First-Order Future Modality). A temporal modality M is a first-order future modality if its truth table is a future formula of *FOMLO*.

Second-order future modalities are defined similarly. The modalities defined in the above example, **Fp**, **U**(p, q) and $F^\infty p$, are first-order future modalities.

Definition 5 (Path Modalities). For every first-order future formula $\varphi(x_0, X_1, \dots, X_l)$, we define an l place path modality $\mathbf{E}\varphi$ as follows: $T, a \models \mathbf{E}\varphi$ if and only if there is a path π from a in T , such that $T_\pi, a \models \varphi(x_0, X_1, \dots, X_l)$. $\mathbf{E}\varphi$ is said to be the path modality which corresponds to $\varphi(x_0, X_1, \dots, X_l)$.

Proposition 6. For every first-order future formula $\varphi(x_0, X_1, \dots, X_l)$, the path modality $\mathbf{E}\varphi$ has an MPL truth table.

3 A Hierarchy

For every $k \geq 1$, let BTL_k be the temporal logic defined as $TL(M_k)$, where $M_k = \{\mathbf{E}\varphi : \text{qd}(\varphi(x_0, X_1, \dots, X_l)) \leq k \text{ and } \varphi \text{ is a first-order future formula}\}$.

Note that for every $k \geq 1$, BTL_k is based on an infinite set of modalities. All the basic modalities of BTL_k have a truth table of quantifier depth $\leq k + 1$. However,

Proposition 7. For every n and $k > 1$ there is a BTL_k formula which is equivalent to no MLO formula of quantifier depth $\leq n$.

We use the following simple property to show that the sequence $\{BTL_k\}_{k=1}^\infty$ contains a true infinite hierarchy.

Definition 8 (Block_k). For $k \geq 1$ let Block_k be a property of trees with two unary predicates p, q , defined as follows. $T \in \text{Block}_k$ iff there is a path π starting at the root of T such that: (1) There is a node $v \in \pi$ such that $v \in q$; (2) Each occurrence of p between the root and v is a part of a sequence of exactly k consecutive p labeled nodes on π ; (3) There are no sequences of $k + 1$ consecutive p labeled nodes on π , between the root and v ; (4) The root of T is labeled p .

Proposition 9. For every $k \geq 1$, there is $k' \geq 1$ such that Block_k is expressible by a formula $\varphi(p, q) \in BTL_{k'}$ (i.e., $T, \text{root} \models \varphi(p, q)$ iff T has property Block_k).

Our main inexpressibility result for BTL_k is:

Theorem 10. For all $k \geq 1$ and $m > 2^k$, there is no BTL_k formula which expresses the property Block_m .

The proof of Th. 10 is based on a new Ehrenfeucht-Fraïssé game on trees, appropriate for BTL_k (see [15] for details). Th. 10 and Proposition 9 implies:

Theorem 11 (Hierarchy). For every $k \geq 1$, exists $k' > k$, such that $BTL_{k'}$ is strictly more expressive than BTL_k .

4 On Temporal Logics over Branching Time

The infinite hierarchy of branching time logics BTL_k defined in section 3, can serve as an external “yardstick” against which other future BTL s can be compared. Below we examine some commonly used branching time logics. These logics are based on different finite and infinite sets of future modalities. Recall that we use $TL(M)$ to indicate the temporal logic based on the set of modalities M . A set of modalities M is a **base** for a temporal logic L if $L \equiv_{exp} TL(M)$. Note that the modalities in M do not have to be basic modalities of L .

4.1 CTL^* Has No Finite Base

The definition of CTL^* [7] uses an interplay between state formulas (which correspond to genuine modalities) and path formulas to generate infinitely many modalities by a finite syntax. Nevertheless, CTL^* can be represented as the temporal logic based on the infinite set of modalities $\{E\varphi : \varphi \text{ is a formula of } TL(U)\} \cup \{A\varphi : \varphi \text{ is a formula of } TL(U)\}$ [5]. $A\varphi$ (respectively $E\varphi$) holds at a current moment t_0 if “for all (respectively, for some) paths from the current moment, φ holds”. Formally, $T, t_0 \models E\varphi$ iff $T_\pi, t_0 \models \varphi$ for some π starting at t_0 . The modalities of the form $A\varphi$ are redundant since for every φ in $TL(U)$, $A\varphi$ means the same as $\neg E\neg\varphi$. Thus, the set of modalities $\{E\varphi : \varphi \in TL(U)\}$ is a base for CTL^* . Kamp’s theorem implies that the set $\{E\varphi : \varphi \in TL(U)\}$ of modalities is semantically equivalent to the set of modalities $\{E\varphi : \varphi \text{ is a future formula of } FOMLO\}$. Therefore:

Lemma 12. $\bigcup_{k=1}^{\infty} BTL_k \equiv_{exp} CTL^*$

Since $\{BTL_k\}_{k=1}^{\infty}$ is a true infinite hierarchy (Theorem 11), we obtain:

Theorem 13. *CTL^* has no finite base.*

In [17], CTL^* was shown to be expressively equivalent to the bisimulation invariant fragment of monadic path logic. Bisimulation equivalence plays a very important role in concurrency. (Recall that $\varphi(x_0)$ is bisimulation invariant if $T, root \models \varphi(x_0)$ implies $T', root' \models \varphi(x_0)$ whenever T and T' are bisimulation equivalent.) Thus, CTL^* represents some objectively quantified expressive power. From the fact that the property $Block_k$ is expressible in $FOMLO$ we obtain the following theorem:

Theorem 14. *The bisimulation invariant fragment of future first-order monadic logic of order has no finite base.*

Hence, the situation for temporal logics over trees (branching time models), is completely different than the situation for temporal logics of linear time, where the temporal logic based on the single modality U is expressively equivalent to the future fragment of first-order monadic logic of order [13]. We believe that this is the reason for the multiplicity of temporal logics over branching time.

4.2 BTL_k vs. Commonly Used Branching Time Logics

Many temporal logics were suggested as branching time specification formalisms (see [8,4]) by imposing some syntactical restrictions on CTL^* formulas. We examine the expressive power of commonly used branching time temporal logics. It turns out that almost all of these logics are inside the second level of our hierarchy. The modalities for these logics were suggested by desire to formalize some pragmatic properties which often occur in specifications of hardware and software systems. It is instructive to observe that most of these properties can be formalized by BTL_2 formulas constructed from basic modalities of quantifier depth two.

In the following list we use the symbols **U**, **F**, **G** to indicate the non-strict versions of the respective temporal operators (see Example 2). $\mathbf{F}^\infty p$ abbreviates $\mathbf{GF}p$ and $\mathbf{G}^\infty p$ abbreviates $\neg \mathbf{F}^\infty \neg p$; its meaning on linear orders is ‘almost everywhere p ’. The meaning of the modality $\mathbf{X}p$ is ‘next time p ’.

$B(\mathbf{F})$ (see [14,4]). Let $M_1 = \{\mathbf{EG}, \mathbf{EF}, \mathbf{AG}, \mathbf{AF}\}$, then $B(\mathbf{F})$ can be defined as $TL(M_1)$. Since the truth tables of **F** and **G** have quantifier-depth = 1 and for every formula p , $\mathbf{AF}p = \neg \mathbf{EG} \neg p$ and $\mathbf{AG}p = \neg \mathbf{EF} \neg p$, $B(\mathbf{F}) \preceq_{exp} BTL_1$. According to [4], the formula $\mathbf{E}(\mathbf{F}p \wedge \mathbf{G}q)$ is not expressible in $B(\mathbf{F})$. Since this formula is expressible in BTL_1 , it follows that $B(\mathbf{F}) \prec_{exp} BTL_1$.

CTL and CTL^+ (see [1]). CTL can be defined as $TL(\mathbf{EX}, \mathbf{AX}, \mathbf{EU}, \mathbf{AU})$. Since the truth table of the **U** operator has quantifier-depth = 2, we have $CTL \preceq_{exp} BTL_2$. Let Φ_1 be the set of $TL(\mathbf{U})$ formulas of nesting depth ≤ 1 . Let M_2 be the infinite set $\{\mathbf{E}\varphi : \varphi \in \Phi_1\}$ of path modalities, then CTL^+ is defined as $TL(M_2)$ and hence $CTL^+ \preceq_{exp} BTL_2$.

$ECTL$ and $ECTL^+$ (see [4]). $ECTL$ and $ECTL^+$ allow specification of fairness properties. $ECTL$ can be defined as $TL(\mathbf{EX}, \mathbf{AX}, \mathbf{EU}, \mathbf{AU}, \mathbf{EF}^\infty, \mathbf{EG}^\infty, \mathbf{AF}^\infty, \mathbf{AG}^\infty)$. Let Φ_2 be the set of $TL(\mathbf{X}, \mathbf{U}, \mathbf{F}^\infty, \mathbf{G}^\infty)$ formulas of nesting depth ≤ 1 . Let M_4 be the infinite set $\{\mathbf{E}\varphi : \varphi \in \Phi_2\}$ of path modalities, then $ECTL^+$ can be defined as $TL(M_4)$. The truth tables of $\mathbf{F}^\infty p$ and $\mathbf{G}^\infty p$, are both of quantifier-depth = 2. Therefore, $ECTL \preceq_{exp} BTL_2$ and $ECTL^+ \preceq_{exp} BTL_2$.

In [7], the formula $\mathbf{AF}(p \wedge \mathbf{X}p)$ was provided as an example for a CTL^* formula which is not expressible in $ECTL^+$. The formula $\mathbf{AF}(p \wedge \mathbf{X}p)$ is expressible in BTL_3 . As far as we know, this is the only modality discussed in the literature which is not definable in BTL_2 .

5 Complexity of Model Checking

The model checking problem for a logic L is as follows. Given a finite Kripke structure K and a formula $\varphi \in L$, determine whether $T_K, root \models \varphi$, where T_K is the tree that corresponds to the unwinding of K from its initial state.

CTL is based on four modalities. The Model Checking (M.C.) problem for CTL has linear time complexity $O(|K| \times |\varphi|)$. CTL^* is based on an infinite

set of modalities. Unlike CTL, the M.C. problem for CTL^* is PSPACE complete [2]. The next Theorem shows that for a temporal logic based on a finite set of modalities, the M.C. problem has low complexity.

Recall that modal μ calculus is equivalent to the bisimulation invariant fragment of (future) monadic second-order logic [12].

Theorem 15 (Complexity of Model Checking). *Let $TL(M_1, M_2, \dots, M_r)$ be a TL based on a finite set of modalities.*

1. *Assume that M_i ($i = 1, 2, \dots, r$) are definable by CTL^* formulas. Then, the M.C. problem for $TL(M_1, M_2, \dots, M_r)$ has time complexity $O(|K| \times |\varphi|)$.*
2. *Assume that M_i ($i = 1, 2, \dots, r$) are of the form $E\psi$, where ψ is a future monadic second-order formula. Then, the M.C. problem for $TL(M_1, M_2, \dots, M_r)$ has time complexity $O(|K| \times |\varphi|)$.*
3. *Assume that M_i ($i = 1, 2, \dots, r$) are definable by μ -formulas. Then, the M.C. problem for $TL(M_1, M_2, \dots, M_r)$ is in PTIME.*

Acknowledgments. We would like to thank Yoram Hirshfeld for his insights that influenced this research.

References

1. E.M. Clarke and E.A. Emerson (1981). Design and verification of synchronous skeletons using branching time temporal logic. *LNCS* **131**:52–71.
2. E.M. Clarke, E.A. Emerson and A.P. Sistla. Automatic verification of finite state concurrent system using temporal logic. In: *POPL*, 1983.
3. A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fundamenta mathematicae* 49,129–141,1961.
4. E.A. Emerson (1990). Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier, Amsterdam, 1990.
5. E.A. Emerson (1996). *Automated Temporal Reasoning about Reactive Systems*. LNCS vol. 1043, pp. 41–101, Springer Verlag 1996.
6. E.A. Emerson and J.Y. Halpern (1982). Decision procedures and expressiveness in the temporal logic of branching time. In *STOC'82*. pp. 169–180.
7. E.A. Emerson and J.Y. Halpern (1986). ‘Sometimes’ and ‘not never’ revisited: on branching versus linear time temporal logics. *Journal of the ACM* **33**(1):151–178.
8. E.A. Emerson and C.L. Lei. Modalities for model checking: branching time strikes back. *12th ACM Symp. on POPL*, pp. 84–96, 1985.
9. D. Gabbay, I. Hodkinson and M. Reynolds (1994). *Temporal Logic*. Oxford University Press.
10. T. Hafer and W. Thomas (1987). Computation tree logic CTL^* and path quantifiers in the monadic theory of the binary tree. In *ICALP'87*, *LNCS* **267**:269–279. Springer-Verlag.
11. Y. Hirshfeld and A. Rabinovich(1999). Quantitative Temporal Logic. *LNCS* **1683**:172–187, Springer-Verlag.
12. D. Janin and I. Walukiewicz (1996). On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR'96*, *LNCS* **1119**:263–277, Springer-Verlag.

13. H.W. Kamp (1968). Tense logic and the theory of linear order. PhD Thesis, University of California, Los Angeles.
14. L. Lamport (1980). "Sometimes" is sometimes "not never" - On the temporal logic of programs. In *POPL'80*. pp. 174-185.
15. S. Maoz (2000). Infinite Hierarchy of Temporal Logics over Branching Time Models. M.Sc. Thesis, Tel-Aviv University.
16. R. Milner (1989). *Communication and Concurrency*. Prentice-Hall.
17. F. Moller and A. Rabinovich (1999). On the expressive power of CTL*. *Proceedings of fourteenth IEEE Symposium on Logic in Computer Science*, 360-369.
18. D.M.R. Park (1981). Concurrency and automata on infinite sequences. *LNCS* **104**:168-183.

Optimal Satisfiability for Propositional Calculi and Constraint Satisfaction Problems

Steffen Reith and Heribert Vollmer

Lehrstuhl für Theoretische Informatik

Universität Würzburg

Am Hubland

D-97074 Würzburg, Germany

[streit,vollmer]@informatik.uni-wuerzburg.de

Abstract. We consider the problems of finding the lexicographically minimal (or maximal) satisfying assignment of propositional formulas for different restricted classes of formulas. It turns out that for each class from our framework, these problems are either polynomial time solvable or complete for OptP.

1 Introduction

In 1978 Thomas J. Schaefer proved a remarkable result. He examined satisfiability of propositional formulas for certain syntactically restricted formula classes. Each such class is given by a set S of Boolean functions allowed when constructing formulas. An S -formula in his sense is a conjunction of clauses, where each clause consists of a Boolean function from S applied to some propositional variables. Such a Boolean function can be interpreted as a constraint that has to be fulfilled by a given assignment; the satisfiability problem for S -formulas hence provides a mathematical model for the examination of the complexity of constraint satisfaction problems. Let $\mathbf{CSP}(S)$ denote the problem to decide for a given S -formula if it is satisfiable. Schaefer showed that, depending on S , the problem $\mathbf{CSP}(S)$ is either (1) efficiently (i.e., in polynomial time) solvable or (2) NP-complete; and he gave a simple criterion that, given S , allows one to determine whether (1) or (2) holds. Since the complexity of $\mathbf{CSP}(S)$ is either easy or hard (and not located in one of the – under the assumption $P \neq NP$ – infinitely many intermediate degrees between P and the NP-complete sets [Lad75]), Schaefer called this a “dichotomy theorem for satisfiability”.

A somewhat more general kind of formulas was investigated by Lewis in 1979 (see [Lew79]). Here we are allowed to build propositional formulas using connectives taken from a finite set B of Boolean functions, instead of the usual connectives $\wedge, \vee, \neg, \dots$. These formulas will be called B -formulas. (To distinguish, S -formulas in Schaefer’s sense will henceforth be referred to as S -CSPs.) Classes of B -formulas are very closely related to closed classes of Boolean functions, which were fully characterized by Post in the twenties of this century (see

[Pos41,JGK70]). Similar to Schaefer, Lewis obtained a dichotomy theorem which states that the satisfiability problem $\mathbf{SAT}(B)$ of such formulas either complete for NP or in P, depending on properties of the set B of allowed functions.

In the last few years, these results regained interest among complexity theorists. Constraint satisfaction problems were studied by Nadia Creignou and others [Cre95,CH96,CH97], see also the monograph [CKS00]. The complexity of problems related to B -circuits and B -formulas was studied in [RW00]. Considering different versions of satisfiability, optimization and counting problems, dichotomy theorems for classes as NP, MaxSNP, PP and $\#P$ were obtained. Also, the study of Schaefer's formulas lead to remarkable results about approximability of optimization problems in the constraint satisfaction context [KST97, KSW97].

In this paper, we continue this line of research by considering the complexity of other optimization problems defined via formula satisfiability, namely the problems LexMax**SAT** and LexMin**SAT** of determining the lexicographically maximal (or minimal) satisfying assignment of a given propositional formula. In the case of unrestricted formulas, these problems are known to be complete for Krentel's class OptP [Kre88]. The main result of the present paper is a clarification of the complexity of the problem to determine maximal or minimal satisfying assignments of formulas given either in the Lewis context by a set of Boolean connectives, or in the Schaefer context, by a set of constraints. We will show that in all cases, the considered problem is either complete for OptP or solvable in polynomial time, depending on the set of allowed connectives or constraints.

After presenting some notations and preliminary results in Sect. 2, we turn to B -formulas in the Post/Lewis-context in Sect. 3. In Sect. 4 we consider constraint satisfaction problems. Finally, Sect. 5 concludes. Proofs of all our results can be found in the full version of this paper available at

<ftp://ftp-info4.informatik.uni-wuerzburg.de/pub/TRs/re-vo00.ps.gz>.

2 Preliminaries

Propositional Logic

Let Φ be a propositional formula. By $\text{Var}(\Phi)$ we denote the set of those variables associated with Φ . Hence, $\text{Var}(\Phi)$ must contain those variables with actual occurrences in Φ , but we also allow $\text{Var}(\Phi)$ to contain additionally so called *fictive* variables. We say that Φ is a *formula over* $\text{Var}(\Phi)$. To denote that Φ is a formula over $\{x_1, \dots, x_k\}$, we also write $\Phi(x_1, \dots, x_k)$. By $\Phi \left[\begin{smallmatrix} x \\ y \end{smallmatrix} \right]$ we indicate the formula created by simultaneously replacing each occurrence of x in Φ by y , where x, y are either variables or constants.

Let V be a finite set of propositional variables. Since we want to compare propositional truth assignments lexicographically, we have to talk about the first, second, etc. variable. Thus we will always assume an ordering on V without further mention.

An *assignment with respect to* V is a function $I: V \rightarrow \{0, 1\}$. When the set V of variables is clear from the context, we will simply speak of an *assignment*. In order for an assignment w.r.t. V to be compatible with a formula Φ , we must have $\text{Var}(\Phi) = V$. That an assignment I satisfies Φ will be denoted by $I \models \Phi$. If I is an assignment w.r.t. V , $y \notin V$, and $a \in \{0, 1\}$, then $I \cup \{y := a\}$ denotes the assignment I' w.r.t. $V \cup \{y\}$, defined by $I'(y) = a$ and $I'(x) = I(x)$ for all $x \neq y$. On the other hand, for $\{x_{i_1}, x_{i_2}, \dots, x_{i_m}\} \subseteq V$, we denote by $I' = I / \{x_{i_1}, x_{i_2}, \dots, x_{i_m}\}$ the assignment I' w.r.t. $\{x_{i_1}, x_{i_2}, \dots, x_{i_m}\}$, given by $I'(x) = I(x)$ iff $x \in \{x_{i_1}, x_{i_2}, \dots, x_{i_m}\}$.

If $V = \{x_1, \dots, x_k\}$, $x_1 < \dots < x_k$, then an assignment I with $I(x_i) = a_i$ will also be denoted by $(a_1, \dots, a_k)$. An ordering on the variables induces an ordering on assignments as follows: $(a_1, \dots, a_k) < (b_1, \dots, b_k)$ if and only if there is an $i \leq k$ such that for all $j < i$ we have $a_j = b_j$ and $a_i < b_i$. We refer to this ordering as the *lexicographical ordering*. We write $(a_1, \dots, a_k) \models_{\min} \Phi$ ($(a_1, \dots, a_k) \models_{\max} \Phi$, resp.) iff $(a_1, \dots, a_k) \models \Phi$ and there exists no lexicographically smaller (larger, resp.) $(a'_1, \dots, a'_k) \in \{0, 1\}^k$ such that $(a'_1, \dots, a'_k) \models \Phi$.

Maximization and Minimization Problems

The study of optimization problems in computational complexity theory started with the work of Krentel [Kre88, Kre92]. We fix the alphabet $\Sigma = \{0, 1\}$. Let FP denote the class of all functions $f: \Sigma^* \rightarrow \Sigma^*$ computable deterministically in polynomial time. Say that a function h belongs to the class MinP if there is a function $f \in \text{FP}$ and a polynomial p such that, for all x , $h(x) = \min_{|y|=p(|x|)} f(x, y)$, where minimization is taken with respect to lexicographical order. The class MaxP is defined by taking the maximum of these values. Finally, let $\text{OptP} = \text{MinP} \cup \text{MaxP}$.

Krentel considered the following reducibility in connection with these classes: A function f is *metric reducible* to h ($f \leq_{\text{met}}^p h$) if there exist two functions $g_1, g_2 \in \text{FP}$ such that for all x we have: $f(x) = g_1(h(g_2(x)), x)$. We say that g_1, g_2 establish the metric reduction from f to h .

Krentel in [Kre88] presented a number of problems complete for OptP under metric reducibility. The for us most important one is the problem of finding the lexicographically minimal satisfying assignment of a given formula, defined as follows:

PROBLEM: LexMin3-SAT

INSTANCE: a propositional formula Φ in 3-CNF

OUTPUT: the lexicographically smallest satisfying assignment of Φ , or “ $\perp$ ” if Φ is unsatisfiable

The problem LexMax3-SAT is defined analogously.

Proposition 1 ([Kre88]). *LexMin3-SAT and LexMax3-SAT are complete for OptP under metric reductions.*

3 Dichotomy Theorems for B -Formulas

3.1 Formulas Given by a Set of Boolean Functions

Any function of the kind $f: \{0,1\}^k \rightarrow \{0,1\}$ will be called (k -ary) *Boolean function*. By BF we denote the set of all Boolean functions. For simplicity we often use propositional formulas to represent Boolean functions, formally: Let Φ be a formula over a set $V = \text{Var}(\Phi)$ of k variables. Then f_Φ , the Boolean function defined (or, *represented*) by Φ , is given by $f_\Phi(a_1, \dots, a_k) = 1$ iff $(a_1, \dots, a_k) \models \Phi$. For example, the functions $\text{id}(x)$, $\text{et}(x, y)$, $\text{vel}(x, y)$, $\text{non}(x)$, $\text{aut}(x, y)$ are represented by the formulas x , $x \wedge y$, $x \vee y$, $\neg x$ and $x \oplus y$. We will use 0 and 1 for the constant 0-ary boolean functions. The variable x_i is called *fictive* in f if $f(a_1, \dots, a_{i-1}, 0, a_{i+1}, \dots, a_n) = f(a_1, \dots, a_{i-1}, 1, a_{i+1}, \dots, a_n)$ for all $a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n$ and $1 \leq i \leq n$.

Let B be a set of Boolean functions. By $[B]$ we denote the smallest set of Boolean functions, which contains B and is closed under superposition, i.e., under substitution (that is, composition of functions), permutation and identification of variables, and introduction of fictive variables. A set F of Boolean functions is a *base* for B if $[F] = B$, and F is called *closed* if $[F] = F$. A base B is called *complete* if $[B] = \text{BF}$. As an example, it is well known that $\{\text{vel}, \text{et}, \text{non}\}$ and $\{\neg(x \wedge y)\}$ are complete bases.

Emil Post [Pos41] gave a complete list of all closed classes of Boolean functions (see also [JGK70]). For us important are the following classes which we introduce by giving their bases.

Base	Class	Base	Class
$\{\text{vel}, x \oplus y \oplus 1\}$	R_1	$\{x \vee \bar{y}\}$	S_0
$\{\text{et}, \text{vel}, 0, 1\}$	M	$\{x \wedge \bar{y}\}$	S_1
$\{\text{aut}, 1\}$	L	$\{x \vee (y \wedge \bar{z})\}$	S_{02}
$\{(x \wedge \bar{y}) \vee (x \wedge \bar{z}) \vee (\bar{y} \wedge \bar{z})\}$	D	$\{x \wedge (y \vee \bar{z})\}$	S_{12}
$\{(x \wedge y) \vee (x \wedge \bar{z}) \vee (y \wedge \bar{z})\}$	D_1		

Now let B be a finite set of Boolean functions. A B -formula Φ is a formula representing a Boolean function $f_\Phi \in [B]$ by describing the substitution structure of f_Φ , where we use fixed symbols for the Boolean functions of B . More precisely, let V be a finite set of variables. Then we define:

- every variable $x \in V$ is a B -formula over V ;
- if $f \in B$ is k -ary, and $\Phi_1, \dots, \Phi_k$ are B -formulas over V , then $\tilde{f}(\Phi_1, \dots, \Phi_k)$ is a B -formula over V , where $\tilde{f}$ is a symbol representing f .

Note that not all variables in V actually have to occur in Φ ; those not appearing are fictive variables.

In the following, we will refer to the satisfiability problem for B -formulas as **SAT**(B). The following result is known:

Proposition 2 ([Lew79]). *If $[B] \supseteq S_1$ then **SAT**(B) is NP-complete, in all other cases, **SAT**(B) $\in$ P.*

3.2 Dichotomy Theorems for LexMinSAT(B) and LexMaxSAT(B)

In this section we study the complexity of finding the lexicographical smallest (largest, resp.) satisfying assignment of B -formulas, for all finite sets B of Boolean functions; formally:

PROBLEM: LexMinSAT(B)
 INSTANCE: a B -formula Φ
 OUTPUT: the lexicographically smallest satisfying assignment of Φ , or “ $\perp$ ” if Φ is unsatisfiable

The corresponding maximization problem is denoted by LexMaxSAT(B).

The cases of formulas with easy LexMin/MaxSAT-problem are easy to identify:

Lemma 3. *If B is a finite set of Boolean functions such that $B \subseteq M$ or $B \subseteq L$, then LexMinSAT(B) $\in$ FP and LexMaxSAT(B) $\in$ FP. If $B \subseteq R_1$, then LexMaxSAT(B) $\in$ FP.*

Our main result in this section rests on the following technical lemma.

Lemma 4. *Let B be a finite set of Boolean functions. If there are B -formulas $E(x, y, v, u)$, $N(x, v, u)$ and $F(v, u, x)$ such that $E(x, y, 0, 1) \equiv x \wedge y$, $N(x, 0, 1) \equiv \neg x$, $F(0, 0, x) \equiv 0$, $F(0, 1, x) \equiv x$ and not $F(1, 0, x) \equiv F(1, 1, x) \equiv 0$, then LexMinSAT(B) is $\leq_{\text{met}}^p$ -complete for MinP.*

For various non complete sets B one can show that formulas E , N , and F , as required in the above lemma, exist.

Theorem 5. *If B is a finite set of functions such that $[B] \supseteq S_{02}$, $[B] \supseteq S_{12}$ or $[B] \supseteq D_1$, then LexMinSAT(B) is $\leq_{\text{met}}^p$ -complete for MinP.*

Combining Lemma 3 and Theorem 5, we are now ready to prove a dichotomy theorem for LexMinSAT(B) for arbitrary finite sets of Boolean functions B :

Corollary 6 (Dichotomy Theorem for LexMinSAT(B)). *Let B be a finite set of Boolean functions. If $[B] \supseteq S_{02}$, $[B] \supseteq S_{12}$ or $[B] \supseteq D_1$, then LexMinSAT(B) is $\leq_{\text{met}}^p$ -complete for MinP. In all other cases LexMinSAT(B) $\in$ FP.*

Proof. The first part of the statement is proved by Theorem 5. Now let B be a finite set of Boolean functions such that $[B] \not\supseteq S_{02}$, $[B] \not\supseteq S_{12}$ and $[B] \not\supseteq D_1$. By Posts results [Pos41] we obtain that either $B \subseteq M$ or $B \subseteq L$. Using Lemma 3 the second part of the statement follows. $\square$

Turning to LexMaxSAT(B), we observe that, if $\text{non} \in [B]$, the problem of finding the minimal satisfying assignment of a B -formula reduces to the problem of determining the maximal satisfying assignment of a B -formula. We will use this to show that LexMaxSAT(B) is hard for OptP if B is the set of selfdual functions.

Lemma 7. *Let B be a finite set of Boolean functions. If $\text{non} \in [B]$, then $\text{LexMinSAT}(B) \leq_{\text{met}}^p \text{LexMaxSAT}(B)$.*

This leads us to the dichotomy theorem for $\text{LexMaxSAT}(B)$.

Corollary 8 (Dichotomy Theorem for $\text{LexMaxSAT}(B)$). *Let B be a finite set of Boolean functions. If $[B] \supseteq S_1$ or $[B] \supseteq D$, then $\text{LexMaxSAT}(B)$ is $\leq_{\text{met}}^p$ -complete for MaxP. In all other cases $\text{LexMaxSAT}(B) \in \text{FP}$.*

Finally, let us remark that a dichotomy result for B -formulas with constants i.e. $0, 1 \in [B]$, is much easier.

Corollary 9. *Let B be a finite set of Boolean functions, such that $[B \cup \{0, 1\}] = \text{BF}$. Then $\text{LexMinSAT}(B \cup \{0, 1\})$ ($\text{LexMaxSAT}(B \cup \{0, 1\})$, resp.) is $\leq_{\text{met}}^p$ -complete for MinP (MaxP, resp.). In all other cases $\text{LexMinSAT}(B \cup \{0, 1\}) \in \text{FP}$ ($\text{LexMaxSAT}(B \cup \{0, 1\}) \in \text{FP}$, resp.).*

4 Constraint Satisfaction Problems

In this section, we will consider formulas that are given as a conjunction of constraints (given by Boolean functions applied to a subset of the variables). At first sight, one might hope that the machinery developed by Post and others is applicable here. However, the “upper-level” conjunction is of a restricted nature, and this does not fit into Post’s definition of (unrestricted) superposition. Informally, Schaefer-like results cannot be obtained mechanically from Post/Lewis-like results, but new proofs are needed.

Dichotomy theorems for the problem to determine minimal satisfying assignments of constraint satisfaction problems with respect to the component-wise order of Boolean vectors were obtained in [KK99]. We here consider lexicographical ordering, as in the previous section.

4.1 Formulas Given by a Set of Constraints

Let S be a set of Boolean functions. In this section we will again assume that such S are nonempty and finite. *S-formulas in the Schaefer sense*, or, *S-CSPs*, will now be propositional formulas consisting of clauses built by using functions from S applied to arbitrary variables. Formally, let $S = \{f_1, f_2, \dots, f_n\}$ be a set of Boolean functions and V be a set of variables. An *S-CSP* Φ (over V) is a finite conjunction of clauses $\Phi = C_1 \wedge \dots \wedge C_k$, where each C_i is of the form $\hat{f}(x_1, \dots, x_k)$, $f \in S$, $\hat{f}$ is the symbol representing f , k is the arity of f , and $x_1, \dots, x_k \in V$. If some variables of an *S-CSP* Φ are replaced by the constants 0 or 1 then this new formula Φ' is called *S-CSP with constants*. If $\Phi = C_1 \wedge \dots \wedge C_k$ is a CSP over V , and I is an assignment with respect to V , then $I \models \Phi$ if Φ satisfies all clauses C_i . Here, a clause $\hat{f}(x_1, \dots, x_k)$ is satisfied, if $f(I(x_1), \dots, I(x_k)) = 1$.

We will consider different types of Boolean functions, following the terminology of Schaefer [Sch78].

- The k -ary Boolean function f is *0-valid* (*1-valid*, resp.) if $f(0^k) = 1$ ($f(1^k) = 1$, resp.).
- The Boolean function f is *Horn* (*anti-Horn*, resp.) if f is represented by a CNF formula having at most one unnegated (negated, resp.) variable in any conjunct.
- A Boolean function f is *bijunctive* if it is represented by a CNF formula having at most two variables in each conjunct.
- The Boolean function f is *affine* if it can be represented by a conjunction of affine functions.

We remark that Schaefer's term 1-valid coincides with Post's 1-reproducing.

A set S of Boolean functions is called 0-valid (1-valid, Horn, anti-Horn, affine, bijunctive, resp.) iff every function in S is 0-valid (1-valid, Horn, anti-Horn, affine, bijunctive, resp.).

The satisfiability problem for S -CSPs (S -CSPs with constants, resp.) is denoted by $\mathbf{CSP}(S)$ ($\mathbf{CSP}_C(S)$, resp.). Schaefer's main result, a dichotomy theorem for satisfiability of constraint satisfaction problems (i.e., propositional formulas of the form "conjunction of a set of constraints"), can be stated as follows:

Proposition 10 (Dichotomy Theorem for Constraint Satisfaction with Constants). *Let S be a set of Boolean functions. If S is Horn, anti-Horn, affine or bijunctive, then $\mathbf{CSP}_C(S)$ is polynomial-time decidable. Otherwise $\mathbf{CSP}_C(S)$ is NP-complete.*

Proposition 11 (Dichotomy Theorem for Constraint Satisfaction). *Let S be a set of Boolean functions. If S is 0-valid, 1-valid, Horn, anti-Horn, affine or bijunctive, then $\mathbf{CSP}(S)$ is polynomial-time decidable. Otherwise $\mathbf{CSP}(S)$ is NP-complete.*

4.2 Dichotomy Theorems for Constraint Satisfaction Problems

The main result of this section is to answer the question for what syntactically restricted classes of formulas, given by a set S of Boolean constraints, Proposition 1 remains valid. For this, we will consider the following problems:

PROBLEM: Lexicographically Minimal CSP ($\text{LexMinCSP}(S)$)
 INSTANCE: an S -CSP Φ
 OUTPUT: the lexicographically smallest satisfying assignment of Φ , or " $\perp$ " if Φ is unsatisfiable

The corresponding problem with constants is denoted by $\text{LexMinCSP}_C(S)$. We also examine the analogous maximization problems $\text{LexMaxCSP}(S)$ and $\text{LexMaxCSP}_C(S)$.

There are known algorithms for deciding satisfiability of CSPs in polynomial time for certain restricted classes of formulas [CH97]. We first observe that these algorithms can easily be modified to find minimal satisfying assignments.

Lemma 12. *Let S be a set of Boolean functions. If S is bijunctive, Horn, anti-Horn or affine, then $\text{LexMinCSP}_C(C), \text{LexMinCSP}(C) \in \text{FP}$. If S is 0-valid, then $\text{LexMinCSP}(S) \in \text{FP}$.*

We remark that, if S is not bijunctive, Horn, anti-Horn and affine, then $\text{LexMinCSP}_C(S)$ cannot be in FP (unless $P = NP$), because Proposition 10 shows that the corresponding decision problem (which is the problem of deciding whether there is *any* satisfying assignment, not necessarily the minimal one) is NP-complete.

An analogous result holds for $\text{LexMinCSP}(S)$, this time relying on Proposition 11. The only case which requires a bit care is that of a 1-valid set S . Hardness here follows a result in [CH97], which shows that the problem to decide if there exists a satisfying assignment which differs from the vector $(1, 1, \dots, 1)$ is NP-complete.

Next we strengthen these observations by showing that if $\text{LexMinCSP}_C(S)$ or $\text{LexMinCSP}(S)$ are not contained in FP, then they are already complete for OptP.

Theorem 13. *Let S be a set of Boolean functions. If S does not fulfill the properties Horn, anti-Horn, bijunctive or affine, then $\text{LexMinCSP}_C(S)$ is $\leq_{\text{met}}^P$ -complete for MinP.*

Mainly we are interested in formulas without constants, so we have to get rid of the constants in the above results. This can be achieved by a suitable reduction.

Theorem 14. *Let S be a set of Boolean functions. If S is not 0-valid, Horn, anti-Horn, bijunctive or affine, then $\text{LexMinCSP}(S)$ is $\leq_{\text{met}}^P$ -complete for MinP.*

Thus we get dichotomy theorems for finding lexicographically minimal satisfying assignments of CSPs, both for the case of formulas with constants and without constants.

Corollary 15 (Dichotomy Theorem for $\text{LexMinCSP}(\cdot)$ with constants). *Let S be a set of Boolean functions. If S is bijunctive, Horn, anti-Horn or affine, then $\text{LexMinCSP}_C(S) \in \text{FP}$. In all other cases $\text{LexMinCSP}_C(S)$ is $\leq_{\text{met}}^P$ -complete for MinP.*

Corollary 16 (Dichotomy Theorem for $\text{LexMinCSP}(\cdot)$). *Let S be a set of Boolean functions. If S is 0-valid, bijunctive, Horn, anti-Horn or affine, then we have $\text{LexMinCSP}(S) \in \text{FP}$. In all other cases $\text{LexMinCSP}(S)$ is $\leq_{\text{met}}^P$ -complete for MinP.*

If we compare the classes of functions in the statements of the above corollaries with those occurring in Schaefer's results (Propositions 10 and 11), we see that $\text{CSP}_C(S)$ is NP-complete if and only if $\text{LexMinCSP}_C(S)$ is MinP complete; and on the other hand, if S is a set of Boolean functions which is 1-valid

but is not 0-valid, Horn, anti-Horn, bijunctive, or affine, then $\mathbf{CSP}(S)$ is in P but $\mathbf{LexMinCSP}(S)$ is MinP complete.

Results analogous to the above for the problem of finding maximal assignments can be proved:

Theorem 17 (Dichotomy Theorem for $\mathbf{LexMaxCSP}(\cdot)$). *Let S be a set of Boolean functions.*

1. *If S is bijunctive, Horn, anti-Horn or affine, then $\mathbf{LexMaxCSP}_C(S) \in \mathbf{FP}$. In all other cases $\mathbf{LexMaxCSP}_C(S)$ is $\leq_{\text{met}}^P$ -complete for MaxP.*
2. *If S is 1-valid, bijunctive, Horn, anti-Horn or affine, then $\mathbf{LexMaxCSP}(S) \in \mathbf{FP}$. Otherwise $\mathbf{LexMaxCSP}(S)$ is $\leq_{\text{met}}^P$ -complete for MaxP.*

5 Conclusion

In this paper we determined the complexity of the problem to compute the lexicographically minimal or maximal satisfying assignment of a given propositional formula for different restricted formula classes. We obtained a number of dichotomy results, showing that these problems are either in FP or OptP-complete.

One might ask if it is not possible to obtain our results about constraint satisfaction problems from the seemingly more general results obtained in Sect. 3. However, as we pointed out at the beginning of Sect. 4, this seems not to be the case. Another hint in that direction is that the results we obtain in the constraint satisfaction context do not speak about closed sets of Boolean functions (the Schaefer classes 0-valid, (anti-)Horn, and bijunctive are not closed in the sense of Post).

It can be seen that we do not need the full power of metric reductions in this paper. In fact, let us define f to be weakly many-one reducible to h if there are functions $g_1, g_2 \in \mathbf{FP}$ where $g_1(z)$ is always a sub-word of z , such that for all x , $f(x) = g_1(h(g_2(x)))$. The step to many-one reductions (where g_1 is the identity) thus is smaller than in the case of metric reductions. It is easy to observe that all the above given completeness results also hold for weak many-one reductions instead of metric reductions. The question that now arises is of course if we can even prove our completeness results for many-one reductions (i.e, always have $g_1(x) = x$). However this cannot be expected for “syntactic” reasons. For example, in Sect. 3, if we use a non-complete base $S_1 \subseteq B \subset \mathbf{BF}$ we have to introduce new variables for using them as a replacement for the constants we need to construct our B -formulas. The assignments to these variables have to be removed later, which means that we need the full power of a weak many-one reduction to do some final manipulation of the value of the function we reduce to.

Acknowledgment. We are grateful to Nadia Creignou, Caen, and Klaus W. Wagner, Würzburg, for a lot of helpful hints.

References

- [CH96] N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125:1–12, 1996.
- [CH97] N. Creignou and J.-J. Hébrard. On generating all solutions of generalized satisfiability problems. *Informatique Théorique et Applications/Theoretical Informatics and Applications*, 31(6):499–511, 1997.
- [CKS00] N. Creignou, S. Khanna, and M. Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. Monographs on Discrete Applied Mathematics. SIAM, 2000. To appear.
- [Cre95] N. Creignou. A dichotomy theorem for maximum generalized satisfiability problems. *Journal of Computer and System Sciences*, 51:511–522, 1995.
- [JGK70] S. W. Jablonski, G. P. Gawrilow, and W. B. Kudrajawzew. *Boolesche Funktionen und Postsche Klassen*. Akademie-Verlag, 1970.
- [KK99] L. Kirousis and P. G. Kolaitis. Dichotomy theorems for minimal satisfiability. manuscript, 1999.
- [Kre88] M. W. Krentel. The complexity of optimization functions. *Journal of Computer and System Sciences*, 36:490–509, 1988.
- [Kre92] M. W. Krentel. Generalizations of OptP to the polynomial hierarchy. *Theoretical Computer Science*, 97:183–198, 1992.
- [KST97] S. Khanna, M. Sudan, and L. Trevisan. Constraint satisfaction: The approximability of minimization problems. In *Proceedings 12th Computational Complexity Conference*, pages 282–296. IEEE Computer Society Press, 1997.
- [KSW97] S. Khanna, M. Sudan, and D. Williamson. A complete classification of the approximability of maximization problems derived from boolean constraint satisfaction. In *Proceedings 29th Symposium on Theory of Computing*, pages 11–20. ACM Press, 1997.
- [Lad75] R. Ladner. On the structure of polynomial-time reducibility. *Journal of the ACM*, 22:155–171, 1975.
- [Lew79] Harry R. Lewis. Satisfiability problems for propositional calculi. *Mathematical Systems Theory*, 13:45–53, 1979.
- [Pos41] E. L. Post. *The Two-Valued Iterative Systems of Mathematical Logic*. Annals of Mathematics Studies 5. Princeton University Press, London, 1941.
- [RW00] Steffen Reith and Klaus W. Wagner. The complexity of problems defined by boolean circuits. Technical Report 255, Institut für Informatik, Universität Würzburg, 2000. To appear in *Proceedings International Conference Mathematical Foundation of Informatics*, Hanoi, October 25–28, 1999.
- [Sch78] T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings 10th Symposium on Theory of Computing*, pages 216–226. ACM Press, 1978.
- [Wag87] K. W. Wagner. More complicated questions about maxima and minima, and some closures of NP. *Theoretical Computer Science*, 51:53–80, 1987.

A Hierarchy Result for Read-Once Branching Programs with Restricted Parity Nondeterminism (Extended Abstract)

Petr Savický^{1,*} and Detlef Sieling^{2,**}

¹ Institute of Computer Science, Academy of Sciences of Czech Republic, Prague,
Czech Republic, email: savicky@cs.cas.cz

² Universität Dortmund, FB Informatik, LS 2, 44221 Dortmund, Germany,
email: sieling@ls2.cs.uni-dortmund.de

Abstract. Restricted branching programs are considered in complexity theory in order to study the space complexity of sequential computations and in applications as a data structure for Boolean functions. In this paper $(\oplus, k)$ -branching programs and $(\vee, k)$ -branching programs are considered, i.e., branching programs starting with a $\oplus$ - (or $\vee$ -) node with a fan-out of k whose successors are k read-once branching programs. This model is motivated by the investigation of the power of nondeterminism in branching programs and of similar variants that have been considered as a data structure. Lower bound methods for these variants of branching programs are presented, which allow to prove even hierarchy results for polynomial size $(\oplus, k)$ - and $(\vee, k)$ -branching programs with respect to k .

1 Introduction

Branching Programs or Binary Decision Diagrams are a well-established model for the representation and manipulation of Boolean functions in computer programs and for the investigation of their complexity. In complexity theory the goal is to prove superpolynomial lower bounds on the size of branching programs for explicitly defined functions, because such lower bounds imply superlogarithmic lower bounds on the sequential space complexity for those functions. Since up to now no method to prove such lower bounds is known, a lot of restricted variants of branching programs have been introduced and proofs of exponential lower bounds for those restricted variants have been presented. The strongest result in this direction is contained in [1]. For further references, see [11] and [15].

Several restricted types of branching programs, in particular OBDDs, which are defined below, are used to represent Boolean functions in applications like hardware design and verification. In such applications, data structures for Boolean functions are needed that allow to store as many important functions as

* Supported by GA CR grant 201/98/0717.

** Supported in part by DFG grant We 1066/9.

possible in small space and to manipulate them efficiently. For more information, see e.g. [4], [15].

In the present paper, we investigate a generalization of read-once branching programs (see below) obtained by combining k read-once branching programs by a parity or a disjunction and prove that it is exponentially more powerful than read-once branching programs for some explicit functions. Moreover, we prove a hierarchy result for these models with respect to k , i.e., we prove for some explicit functions that the size may decrease from exponential to polynomial if k is increased by 1. This result holds for $k \leq (1/3) \log^{1/2} n$.

Let $X = \{x_0, \dots, x_{n-1}\}$ be a set of Boolean variables. A deterministic branching program over X is a directed acyclic graph. The graph consists of sink nodes without outgoing edges and of internal nodes with a fan-out of 2. Each sink is labeled by $c \in \{0, 1\}$. Each internal node v is labeled by a variable from X and has an outgoing 0-edge and an outgoing 1-edge. Furthermore the branching program has a source node, i.e., a node without incoming edges. The function represented by the branching program is evaluated in the following way: For some input $a = (a_0, \dots, a_{n-1})$ the evaluation starts at the source. At each internal node v labeled by x_i the computation proceeds to the successor of v that is reached via the a_i -edge leaving v . The label of the sink that is finally reached is equal to value of the represented function on the input a . The path that is followed for the input a is called the computation path for a .

In a read-once branching program on each path from the source to a sink each variable may be tested at most once. An OBDD (Ordered Binary Decision Diagram) is a read-once branching program where an ordering of the variables is fixed and during each computation the variables are tested according to this ordering. OBDDs have been proposed by Bryant [4] as a data structure for the representation and manipulation of Boolean functions.

A nondeterministic read-once branching program may contain “guessing” nodes, i.e., nodes not labeled by any variable and with an arbitrary number of outgoing edges. Then there may be multiple computation paths for the same input, and an input is accepted, i.e. the value of the represented function is 1, if and only if there is an accepting path for it, i.e., a path leading to the 1-sink. A parity read-once branching program is a nondeterministic read-once branching program with the parity acceptance mode, i.e., an input is accepted, iff there is an odd number of accepting paths for it. It is possible to determine whether an input is accepted or not in time linear in the number of edges by a simple bottom-up evaluation procedure. For more details on the different variants of nondeterminism in branching programs we refer to Meinel [10].

Exponential lower bounds for deterministic and nondeterministic read-once branching programs are known for a long time, see e.g. [14], [16], [6], [3] and [9]. On the other hand, the problem of proving superpolynomial lower bounds for parity read-once branching programs is still open. As a step towards proving lower bounds for parity read-once branching programs Krause [8] proved lower bounds for *oblivious* parity branching programs with bounded length. A bran-

ching program is called oblivious of length l , if its node set can be partitioned into l levels such that all nodes of each level are labeled by the same variable.

The only fact known about (nonoblivious) parity read-once branching programs is that the set of functions with polynomial size parity read-once branching programs is different from the set of functions with polynomial size nondeterministic read-once branching programs. The former class is closed under complement while the latter is not (Krause, Meinel and Waack [9]).

Our main result is the proof of exponential lower bounds for read-once branching programs with restricted parity nondeterminism. We shall consider $(\oplus, k)$ -branching programs. The source of such a branching program is a nondeterministic node (labeled by $\oplus$) with a fan-out of k . The k successors of the source are deterministic read-once branching programs $P_1, \dots, P_k$. The semantics of such a branching program is defined in a straightforward way: It computes the value 1 for some input a iff an odd number of the read-once branching programs $P_1, \dots, P_k$ compute the value 1 for a . Similarly, we define $(\vee, k)$ -branching programs. Now the source is a node (labeled by $\vee$) with a fan-out of k , where the k outgoing edges point to deterministic read-once branching programs $P_1, \dots, P_k$. The value 1 is computed for the input a if at least one of the branching programs $P_1, \dots, P_k$ computes a 1 for a .

We shall prove exponential lower bounds and a hierarchy result for $(\oplus, k)$ -branching programs and $(\vee, k)$ -branching programs. To the best of our knowledge for $(\oplus, k)$ -branching programs exponential lower bounds were not known before. Hierarchy result means that we present a function with polynomial size $(\oplus, k+1)$ -branching programs but only exponential size $(\oplus, k)$ -branching programs (and a different function proving a similar statement for $(\vee, k)$ -branching programs). By de Morgan's rules the hierarchy result for $(\vee, k)$ -branching programs implies a similar hierarchy result for $(\wedge, k)$ -branching programs. We see that increasing the amount of nondeterminism only slightly increases the computational power of polynomial size nondeterministic branching programs. The results of Jukna [6], Krause, Meinel and Waack [9] and Borodin, Razborov and Smolensky [3] imply exponential lower bounds for $(\vee, k)$ -branching programs. In the case of $(\vee, k)$ -branching programs, the contribution of the present paper is the hierarchy result.

The idea to restrict the amount of nondeterminism in branching programs by restricting nondeterminism to the source and bounding the outdegree of the source was also inspired by the hierarchy results for $(\vee, k)$ -OBDDs due to Bollig and Wegener [2] and Sauerhoff [12]. A $(\vee, k)$ -OBDD is a branching program with a $\vee$ -node at the source with k outgoing edges pointing to OBDDs $P_1, \dots, P_k$ (with possibly different variable orderings). The motivation to consider $(\vee, k)$ -OBDDs was given by Jain, Bitner, Fussell and Abraham [5] who suggested to use so-called Partitioned BDDs, which are in fact restricted $(\vee, k)$ -OBDDs, as a data structure for Boolean functions. Another work considering restricted nondeterminism is the due to Sauerhoff [13]. He shows that restricting nondeterminism to the source of a nondeterministic OBDD may cause an exponential blow-up of the size compared with ordinary nondeterministic OBDDs.

The paper is organized as follows. In the following section we describe the general lower bound methods for $(\oplus, k)$ - and $(\vee, k)$ -branching programs. In Section 3 we present the definitions of the functions separating the hierarchies, and we prove the hierarchy results.

2 The Lower Bound Method

We first describe the lower bound method for $(\oplus, k)$ -branching programs. The method is applicable to all (m, k) -full-degree functions, defined in Definitions 1 and 2. The lower bound for such functions is stated in Theorem 3. At the end of this section we show how to adapt this lower bound method to $(\vee, k)$ -branching programs. The lower bound is shown for (m, k) -full-sensitive functions (Definition 8, Theorem 9). In the following let $X = \{x_0, \dots, x_{n-1}\}$ denote the set of variables.

Definition 1. Let $A \subseteq X$. A mapping $\phi : \{0, 1\}^d \rightarrow \{0, 1\}^A$ is called a projection of degree d , if each of the $|A|$ coordinates of $\phi(y_1, \dots, y_d)$ is defined by a literal in one of the variables y_i , $i = 1, \dots, d$, or a constant and, moreover, each of the variables $y_1, \dots, y_d$ is used (positively or negatively) at least once.

Definition 2. A Boolean function f is called (m, k) -full-degree, if the following is satisfied. For any partition of its variables into subsets A, B , where $|A| \leq m$, and every projection $\phi : \{0, 1\}^d \rightarrow \{0, 1\}^A$ of degree $d \leq k$, there is a setting b to the variables B , such that substituting $\phi(y_1, \dots, y_d)$ for the variables in A and b for the variables in B leads to a function $f(\phi(y_1, \dots, y_d), b)$, which is an $\mathbb{F}_2$ -polynomial of degree d in the variables $y_1, \dots, y_d$.

Let us point out that the (m, k) -full-degree property generalizes the m -mixed property introduced by Jukna [6], since a function is m -mixed if and only if it is $(m, 1)$ -full-degree. In the last section, the following theorem is used in situations, where $k = \Theta(\log n)$ and $m/k^{2^k} = \Omega(n^{1/2-\epsilon})$.

Theorem 3. *If a Boolean function f of n variables is (m, k) -full-degree, then each $(\oplus, k)$ -branching program for f has at least $2^{\Omega(m/k^{2^k}) - \log n}$ nodes.*

Proof. Let f be (m, k) -full-degree, and let a $(\oplus, k)$ -branching program P for f be given. Let P consist of the read-once branching programs $P_1, \dots, P_k$. In the following we assume that $P_1, \dots, P_k$ are complete read-once branching programs (i.e., on each computation path each variable is tested exactly once). Since making read-once branching programs complete increases the size by a factor of at most $O(n)$, the lower bound $2^{m/k^{2^k}-1}$ on the total size of the complete branching program, which we prove in the following, implies the claimed lower bound.

Let $t = \lfloor m/k \rfloor$, and for $i \in \{1, \dots, k\}$ let V_i be the set of all nodes on the $(t+1)$ -th level of P_i (the nodes that are reached after t tests have been performed). For every input a k -tuple $(v_1, \dots, v_k) \in V_1 \times \dots \times V_k$ of nodes is reached. Now, let $(v_1, \dots, v_k)$ be fixed. Since the read-once branching programs

$P_1, \dots, P_k$ are complete, on each path from the source of P_i to v_i the same set X_i of variables is tested. Let $A = \bigcup_{i=1}^k X_i$ and let $B = X - A$. By the choice of v_i we have $|X_i| = t$ and $|A| \leq m$. Let T be the set of all settings of the variables in A for which $v_1, \dots, v_k$ are reached. We are going to prove the upper bound $2^{|A|+1}/2^{t/4^k}$ on the size of T . Any upper bound U on the size of T implies the lower bound $2^{|A|}/U$ on the number of tuples $(v_1, \dots, v_k)$. Since the total size of the branching program is at least the k th root of the number of such tuples, the claimed lower bound follows.

Let us remark that after reaching v_i , the branching program P_i “forgets” the information, which of the settings in T is consistent with the input, and this information cannot be recovered any more. We show that if T is large enough, then it contains a subset of size at most 2^k for which this fact prevents the branching program from computing any (m, k) -full-degree function. The critical subset used for this is an image of an appropriate projection with the following property.

Definition 4. A projection $\phi : \{0, 1\}^d \rightarrow \{0, 1\}^A$ is called a covering projection if for every $i = 1, \dots, k$, there is a variable among $y_1, \dots, y_d$ such that all its occurrences (negative and positive) are only used to determine the values of X_i -variables in the output of ϕ .

We split the proof of the upper bound on the size of T into two lemmas. If $|T|$ is large, the first lemma guarantees the existence of a suitable covering projection. By the second lemma, this implies that the computed function is not an (m, k) -full-degree function in contradiction to the assumptions of the theorem. Hence, the two lemmas imply the upper bound $2^{|A|+1}/2^{t/4^k}$ on $|T|$ and complete the proof of Theorem 3.

Lemma 5. *If $|T| \geq 2^{|A|+1}/2^{t/2^{2k}}$, there is a covering projection ϕ such that the degree d of ϕ is at most k and $\phi(\{0, 1\}^d) \subseteq T$.*

Lemma 6. *Let ϕ be a covering projection of degree $d \leq k$, and let $\phi(\{0, 1\}^d) \subseteq T$. For each setting b of the variables in B the followings holds: If in P the variables in A are substituted by $\phi(y_1, \dots, y_d)$ and the variables in B are substituted by b , the represented function is a polynomial of degree at most $d - 1$ over $y_1, \dots, y_d$.*

Proof of Lemma 6. We first consider the effect of substituting the variables in A by $\phi(y)$ and the variables in B by b on the function represented by the read-once branching program P_i . Let $P_i(\phi(y), b)$ denote the result of the substitution. All the variables tested on paths from the source to v_i belong to A . Since $\phi(\{0, 1\}^d) \subseteq T$, for each setting of the y -variables the computation of P_i goes through the node v_i . Let y_j be the variable whose occurrences in ϕ only determine X_i -variables. Then the computation of $P_i(\phi(y), b)$ does not test the variable y_j at v_i or after v_i , i.e., the function computed at v_i does not essentially depend on y_j . It follows that the function computed by P_i is a polynomial of degree at most $d - 1$. Then also the function represented by P is a polynomial of degree at most $d - 1$, since it is the parity of the functions represented by the P_i . $\square$

Proof of Lemma 5. Let $d \leq k$ and let $A_1, \dots, A_{d+1}$ be a partition of A such that for each X_i there is a set $A_{j(i)}$, $j(i) \leq d$, such that $A_{j(i)} \subseteq X_i$. We discuss the choice of this partition later on. We are going to construct a covering projection by considering special rectangular sets. Let $s \in \{0, \dots, d\}$ and let $D_s = \mathcal{A}_1^{(2)} \times \dots \times \mathcal{A}_s^{(2)} \times \mathcal{A}_{s+1} \times \dots \times \mathcal{A}_{d+1}$, where $\mathcal{A}_i$ is the set of all settings of the variables of A_i , and $\mathcal{A}_i^{(2)}$ is the set of all unordered pairs of such settings. The elements of D_s are $(d+1)$ -tuples of the form $(\{a_1, b_1\}, \dots, \{a_s, b_s\}, \{a_{s+1}\}, \dots, \{a_{d+1}\})$, where $a_i, b_i \in \{0, 1\}^{A_i}$ and $a_i \neq b_i$ for $1 \leq i \leq s$, and $a_i \in \{0, 1\}^{A_i}$ for $s+1 \leq i \leq d+1$. We interpret each element of D_s as the product $\{a_1, b_1\} \times \dots \times \{a_s, b_s\} \times \{a_{s+1}\} \times \dots \times \{a_{d+1}\}$, which is a set of 2^s settings of the variables in A . We call such sets *rectangular sets* of dimension s .

We may consider elements of T as rectangular sets of dimension 0, i.e. $T \subseteq D_0$. For any $0 \leq s \leq d$, let $T_s \subseteq D_s$ be the set of all rectangular sets of dimension s that are subsets of T . In particular, $T_0 = T$. We shall prove that T_d is not empty, provided that all sets A_i , $i = 1, \dots, d$, are large enough. Then there is an element $(\{a_1, b_1\}, \dots, \{a_d, b_d\}, \{a_{d+1}\})$ of T_d . Let ϕ be the projection defined by

$$\phi(y_1, \dots, y_d) = (c_1, \dots, c_d, a_{d+1}), \text{ where } c_i = \begin{cases} a_i, & \text{if } y_i = 0, \\ b_i, & \text{if } y_i = 1. \end{cases}$$

The choice of the partition $A_1, \dots, A_{d+1}$ implies that ϕ is a covering projection. Since ϕ is constructed from a rectangular set in T_d , we have $\phi(\{0, 1\}^d) \subseteq T$.

It remains to prove that for a suitable choice of the partition $A_1, \dots, A_{d+1}$ the set T_d is not empty. Let $\text{density}(T_s) = |T_s|/|D_s|$. The following lemma shows how to obtain lower bounds on the density of T_{s+1} from a lower bound on the density of T_s . By applying this lemma inductively, one can obtain that the density of T_d is larger than 0, i.e., that T_d is not empty.

Lemma 7. *Let $s \in \{0, \dots, d-1\}$, let $a = |\mathcal{A}_{s+1}|$ and let $\varepsilon = \text{density}(T_s)$. Then*

$$\text{density}(T_{s+1}) \geq \varepsilon^2 \left(1 - \frac{1}{\varepsilon a}\right).$$

Proof. Partition $D_s = \mathcal{A}_1^{(2)} \times \dots \times \mathcal{A}_s^{(2)} \times \mathcal{A}_{s+1} \times \dots \times \mathcal{A}_{d+1}$ into classes of elements that coincide in all coordinates except the $(s+1)$ -th one. Each of these classes has size $a = |\mathcal{A}_{s+1}|$. Let $N = |D_s|/a$ be the number of these classes and let l_i for $i = 1, 2, \dots, N$ be the size of the intersection of T_s and the i th class. Clearly, $(1/N) \sum_{i=1}^N l_i = |T_s|/N = \varepsilon a$. Since there are $\binom{l_i}{2}$ pairs of elements of the i th class, we obtain from the i th class $\binom{l_i}{2}$ elements of T_{s+1} . Furthermore, the size of D_{s+1} is $N \binom{a}{2}$. Hence, we have the estimate

$$\text{density}(T_{s+1}) = \frac{1}{N \binom{a}{2}} \sum_{i=1}^N \binom{l_i}{2} \geq \frac{1}{\binom{a}{2}} \binom{\varepsilon a}{2} \geq \frac{\varepsilon a (\varepsilon a - 1)}{a^2} = \varepsilon^2 \left(1 - \frac{1}{\varepsilon a}\right),$$

where the first inequality follows from the convexity of $\binom{x}{2}$. □

Since the set X_i contains at most 2^{k-1} cells of the Venn diagram of the sets X_j , $j \neq i$, we may choose for each set X_i a cell contained in X_i of size at least $|X_i|/2^{k-1} = t/2^{k-1}$. In this way we may obtain $d \leq k$ disjoint subsets $A_1, \dots, A_d$ of A of size at least $t/2^k$, such that for each X_i there is a set $A_{j(i)}$ among $A_1, \dots, A_d$ such that $A_{j(i)} \subseteq X_i$. Let $A_{d+1} = A - (A_1 \cup \dots \cup A_d)$. Since we apply Lemma 7 only for $s \in \{0, \dots, d-1\}$, in all applications of the lemma we have $a = 2^{|A_{s+1}|} \geq 2^{t/2^k}$.

Let ε_0 be the density of T_0 ($= T$). By the assumption of Lemma 5 we have $|T| \geq 2^{|A|+1}/2^{t/2^{2k}}$ and, therefore, $\varepsilon_0 \geq \frac{2}{2^{t/2^{2k}}}$. Let ε_s be the lower bound on the density of T_s that we obtain after the s th application of Lemma 7. Clearly, $\varepsilon_0 a \geq 2 \cdot 2^{t/2^k - t/2^{2k}} \geq 2$. Hence, the first application of Lemma 7 yields $\text{density}(T_1) \geq \varepsilon_1 \geq \varepsilon_0^2/2 = 2(\varepsilon_0/2)^2$. It is easy to verify that $\varepsilon_1 a \geq 2$ and we can estimate the density after the second application of the lemma in a similar way. In general, after the s th application of the lemma, we obtain $\text{density}(T_s) \geq \varepsilon_s \geq 2(\varepsilon_0/2)^{2^s}$. For every $s < d$, we have $\varepsilon_s a \geq 2 \cdot 2^{t/2^k - t/2^{2^{k-s}}} \geq 2$, which allows to perform the next step. Hence, after d applications of Lemma 7 we obtain a positive lower bound on the density of T_d , which implies the existence of a covering projection. $\square$

Finally, we present without proof the adaptation of the lower bound method to $(\vee, k)$ -branching programs. The lower bound method can be applied to functions that are (m, k) -full-sensitive—a property that is defined in the following definition.

Definition 8. A function g on d variables is called full-sensitive, if there is an input c for g such that $g(c) = 1$ and the shortest prime implicant covering c has length d . A function f is called (m, k) -full-sensitive, if the following is satisfied. For any partition of its variables into subsets A, B , where $|A| \leq m$, and every projection $\phi : \{0, 1\}^d \rightarrow \{0, 1\}^A$ of degree $d \leq k$, there is a setting b to the variables B such that substituting $\phi(y_1, \dots, y_d)$ for the variables in A and b for the variables in B leads to a full-sensitive function $f(\phi(y_1, \dots, y_d), b)$.

Theorem 9. If a Boolean function f of n variables is (m, k) -full-sensitive, then each $(\vee, k)$ -branching program for f has at least $2^{\Omega(m/k^2 4^k) - \log n}$ nodes.

3 The Lower Bound and the Hierarchy Results

We start with the definitions of the functions separating the classes of the hierarchies. The definitions for these functions are a bit involved since we have simultaneously to prove exponential lower bounds for $(\oplus, k)$ - or $(\vee, k)$ -branching programs and polynomial upper bounds for $(\oplus, k+1)$ - or $(\vee, k+1)$ -branching programs, resp. We remark that for merely proving exponential lower bounds simpler functions can be considered, which even lead to slightly larger lower bounds.

The considered functions are multipointer functions where the pointers are obtained similarly to the matrix storage access function due to Jukna, Razborov, Savický and Wegener [7]. We first describe how to compute a single pointer. Let n be a power of 2. The input $X = \{x_0, \dots, x_{n-1}\}$ is partitioned into blocks. Each block B consists of $\log n$ matrices of size $p \times p$ where p will be chosen later on. The pointer corresponding to B is obtained in the following way: The i th bit s_i of the pointer takes the value 1 iff the i th matrix of B contains a row only consisting of ones, and otherwise s_i takes the value 0. The pointer corresponding to B is $(s_{\log n-1}, \dots, s_0)$ interpreted as a binary number.

In order to compute $f_n^k(x_0, \dots, x_{n-1})$ the input bits $x_0, \dots, x_{n-1}$ are partitioned into $(k+1)k$ blocks $B_{i,j}$, where $i \in \{1, \dots, k+1\}$ and $j \in \{1, \dots, k\}$. Then

$$p = \left\lfloor \left(\frac{n}{k(k+1) \log n} \right)^{1/2} \right\rfloor$$

is a suitable choice such that each bit of the input is contained in at most one matrix. Let $b_{i,j}$ be the pointer obtained from $B_{i,j}$ as described above. Then $f_n^k(x)$ takes the value 1 iff

1. $\forall j \in \{1, \dots, k\} : b_{1,j} = \dots = b_{k+1,j}$ and
2. $x_{b_{1,1}} \wedge \dots \wedge x_{b_{1,k}} = 1$.

For the definition of the function g^k we use exactly the same notation. The function $g_n^k(x)$ takes the value 1 iff

1. $\forall j \in \{1, \dots, k\} : b_{1,j} = \dots = b_{k+1,j}$ and
2. $x_{b_{1,1}} \oplus \dots \oplus x_{b_{1,k}} = 1$.

The only difference to f is that in condition 2. the $\wedge$ is replaced by a $\oplus$.

Informally, X is partitioned into $k+1$ sectors each consisting of k blocks. In each sector there is a block for each of the k pointers. The function may take a value different from 0 only if the sequences of the k pointers for all sectors coincide. Note that the fact that two pointers coincide does not imply that the blocks from which the pointers are derived are identical. The lower and upper bound results are proved in the following theorems.

Theorem 10. *There are $(\oplus, k+1)$ -branching programs for the function f^k and $(\vee, k+1)$ -branching programs for the function g^k of size $O(kn^{k+1})$. These branching programs even consist of $k+1$ OBDDs.*

Proof. We start with the construction of a $(\oplus, k+1)$ -branching program P for f^k . We describe the OBDDs P_i , $i \in \{1, \dots, k+1\}$, that P consists of. The OBDD P_i works in the following way. It first reads the i th sector of the input and stores for each matrix whether it contains a row only consisting of ones. Since there are $k \log n$ matrices in the i th sector, width $O(n^k)$ is sufficient. In particular, after reading the i th sector all pointers of the i th sector are known.

If there is a pointer addressing a bit of the i th sector, P_i computes a 0. If there is some sector $j < i$ such that no pointer addresses any bit of the j th sector, also

a 0 is computed. Otherwise i is the smallest number such that there is no pointer addressing a bit in the i th sector. Then P_i sequentially reads the other sectors and compares the stored pointers of the i th block with the corresponding pointers of the other blocks in order to test condition 1. Since the pointers are stored, it is possible to compute the conjunction of the addressed bits during the comparison of the pointers. The correctness of P follows from the observation that exactly one of the branching programs P_i , namely that where i is the smallest number of a sector without an addressed bit, computes the correct function value, while all P_j , where $j \neq i$, compute a 0. The branching program P_i is able to compute the function value since it has not read any of the addressed bits before it knows all pointers.

The total size of P is bounded by $O(kn^{k+1})$. For the function g^k and $(\vee, k+1)$ -branching programs the same arguments work with the only exception that the parity of the addressed bit has to be computed instead of the conjunction. This may increase the width by a factor of at most 2. $\square$

Theorem 11. *Each $(\oplus, k)$ -branching program for f^k has at least*

$$2^{\Omega\left(\frac{n^{1/2}}{k^{3/4} \log^{1/2} n}\right) - \log n}$$

nodes. This number grows exponentially, if $k \leq (1/4 - \gamma) \log n$ for some $\gamma > 0$.

Proof. By Theorem 3 it suffices to prove that f^k is $(p-1, k)$ -full-degree. Let $A \subseteq X$ such that $|A| \leq p-1$. Let $\phi: \{0, 1\}^d \rightarrow \{0, 1\}^A$ be a projection of degree $d \leq k$. Since by the definition of the projections each variable $y_1, \dots, y_d$ occurs at least once in the projection, we can define $s(1), \dots, s(d)$ in such a way that $x_{s(i)}$ is an occurrence of y_i . Furthermore, let $s(d+1), \dots, s(k)$ be equal to $s(d)$. It follows easily from the definition of f^k that we can choose a setting b of the variables in $B = X - A$ in such a way that in each sector the pointers take the values $s(1), \dots, s(k)$ independent of the setting of the variables in A . The resulting function $f^k(\phi(y), b)$ is the conjunction of the y -variables, which is an $\mathbf{F}_2$ -polynomial of degree d . $\square$

Theorem 12. *Each $(\vee, k)$ -branching program for g^k has at least*

$$2^{\Omega\left(\frac{n^{1/2}}{k^{3/4} \log^{1/2} n}\right) - \log n}$$

nodes. This number grows exponentially, if $k \leq (1/4 - \gamma) \log n$ for some $\gamma > 0$.

We omit the proof of this theorem. In order to state the hierarchy result, let $P-(\oplus, k)$ -BP denote the set of all Boolean functions with polynomial size $(\oplus, k)$ -branching programs, and let $P-(\vee, k)$ -BP and $P-(\wedge, k)$ -BP be defined similarly.

Theorem 13. *If $k \leq (1/2 - \gamma) \log^{1/2} n$ for some $\gamma > 0$, it holds that*

$$\begin{aligned} P-(\oplus, k)\text{-BP} &\subsetneq P-(\oplus, k+1)\text{-BP}, \\ P-(\vee, k)\text{-BP} &\subsetneq P-(\vee, k+1)\text{-BP} \text{ and} \\ P-(\wedge, k)\text{-BP} &\subsetneq P-(\wedge, k+1)\text{-BP}. \end{aligned}$$

Sketch of Proof. The third inequality follows from the second one by de Morgan's rules. For constant k the first and second inequalities follow directly from Theorems 10–12. The hierarchy results for nonconstant k can be proved by a padding argument. $\square$

References

1. Ajtai, M. (1999). A non-linear time lower bound for Boolean branching programs. In *Proc. of 40th Symposium on Foundations of Computer Science*, 60–70.
2. Bollig, B. and Wegener, I. (1999). Complexity theoretical results on partitioned (nondeterministic) binary decision diagrams. *Theory of Computing Systems* 32, 487–503.
3. Borodin, A., Razborov, A. and Smolensky, R. (1993). On lower bounds for read- k -times branching programs. *Computational Complexity* 3, 1–18.
4. Bryant, R.E. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* 35, 677–691.
5. Jain, J., Bitner, J., Fussell, D.S. and Abraham, J.A. (1992). Functional partitioning for verification and related problems. *Brown MIT VLSI Conference*, 210–226.
6. Jukna, S. (1988). Entropy of contact circuits and lower bounds on their complexity. *Theoretical Computer Science* 57, 113–129.
7. Jukna, S., Razborov, A., Savický, P. and Wegener, I. (1997). On P versus $NP \cap co-NP$ for decision trees and read-once branching programs. In *Proc. of Mathematical Foundations of Computer Science*, Springer, Lecture Notes in Computer Science 1295, 319–326.
8. Krause, M. (1990). Separating $\oplus L$ from L , NL , $co-NL$ and $AL (= P)$ for oblivious Turing machines of linear access time. In *Proc. of Mathematical Foundations of Computer Science*, Springer, Lecture Notes in Computer Science 452, 385–391.
9. Krause, M., Meinel, C. and Waack, S. (1991). Separating the eraser Turing machine classes L_e , NL_e , $co-NL_e$ and P_e . *Theoretical Computer Science* 86, 267–275.
10. Meinel, C. (1990). Polynomial size Ω -branching programs and their computational power. *Information and Computation* 85, 163–182.
11. Razborov, A. A. (1991). Lower bounds for deterministic and nondeterministic branching programs. In *Proc. of Fundamentals in Computing Theory*, Springer, Lecture Notes in Computer Science 529, 47–60.
12. Sauerhoff, M. (1999). An improved hierarchy result for partitioned BDDs. To appear in *Theory of Computing Systems*.
13. Sauerhoff, M. (1999). Computing with restricted nondeterminism: the dependence of the OBDD size on the number of nondeterministic variables. In *Proc. of 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, Springer, Lecture Notes in Computer Science 1738, 342–355.
14. Wegener, I. (1988). On the complexity of branching programs and decision trees for clique functions. *Journal of the Association for Computing Machinery* 35, 461–471.
15. Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams—Theory and Applications*. SIAM Monographs on Discrete Mathematics and Its Applications, in print.
16. Žák, S. (1984). An exponential lower bound for one-time-only branching programs. In *Proc. of Mathematical Foundations of Computer Science*, Springer, Lecture Notes in Computer Science 176, 562–566.

On Diving in Trees

Thomas Schwentick

Johannes Gutenberg-Universität Mainz

Abstract. The paper is concerned with queries on tree-structured data. It defines fragments of first-order logic (FO) and FO extended by regular expressions along paths. These fragments have the same expressive power as the full logics themselves. On the other hand, they can be evaluated reasonably efficient, even if the formula which represents the query is considered as part of the input.

1 Introduction

In recent years the concept of semistructured data has been of steadily growing relevance. It is located at the meeting point of developments in various research areas such as databases, structured document processing and the web. The language XML is becoming a universal description format for all kinds of data.

Many query languages have been developed for XML, see, e.g., [4,1]. Most of them combine the query primitives that are used in relational databases, hence SQL, with constructs that allow to navigate along paths, often by means of regular expressions. Given that both SQL and regular expressions have clean logical foundations, by first-order (FO) logic and monadic second-order logic (MSO), respectively, it is an obvious question whether these logics can be combined to a similarly clean logical foundation of query languages for semistructured data.

Semistructured data can be viewed in terms of trees which might carry values on edges as well as in vertices. Such trees are unranked, i.e., there is no a priori limit on the number of children of a vertex. Furthermore, the children of a vertex are ordered. General graphs can be modeled in this framework by encoding additional edges by vertex values, e.g., by means of object identifiers.

Queries on such structures can be conceptionally separated into two classes. One class contains queries that make use of comparisons of values of different vertices. E.g., such a query might ask which names of employees appear at least twice in a document. The other class consists of queries that do not use such comparisons. We refer to them as *join-free* queries. They are very common in the literature, especially in the context of structured documents. They deserve special attention as they allow a simplified modelling which in turn permits very efficient evaluation algorithms via the logic-automaton connection. In this paper we will exclusively investigate join-free queries.

As a very simple example of a join-free query consider *select all employees with last name "Jones"*. To evaluate this query the tree can be interpreted as having a unary attribute *Jones* which holds for every vertex which contains the

string *Jones*. This perspective allows the modelling of the data tree as a finite structure in the sense of mathematical logic.

It has been well-known for a long time that MSO sentences on (ranked) trees can be evaluated in linear time, if the query is considered fixed [12,2]. In more recent work (e.g., [10,7,8]) it is shown that also on unranked trees fixed unary MSO queries can be evaluated in linear time in the size of the tree. If the query is considered as part of the input (i.e., in *combined complexity*) the situation is less desirable as the translation of an MSO formula into a tree automaton has non-elementary complexity. For the combined complexity of evaluating MSO formulas on trees no feasible algorithm seems to be known.

In this context, in [9] a fragment of MSO logic was defined which allows to specify all unary MSO queries with query evaluation in time $O(|\text{tree}|2^{|\text{formula}|})$. Furthermore, this paper introduced, as an intermediate step between FO and MSO logic, an extension (called FOREG, here) of FO by regular expressions along vertical and horizontal paths.

The present paper defines corresponding fragments of FO and FOREG, named GFO and GFOREG, and proves that they can express all FO and FOREG queries, respectively. Furthermore, they inherit efficient evaluation algorithms from [9]. This first result is restricted to unary queries. As a second contribution, we show that, queries of arbitrary arity can be obtained by using only unary queries that are combined by means of regular path expressions.

The result that GFO is equivalent to FO is similar to the result in [5] that the temporal logic CTL* coincides with FO on trees. It uses a decomposition technique which has been used in several papers, e.g., [13,5].

The paper is organized as follows. After some preliminary definitions in Section 2 we introduce GFO and GFOREG in Section 3 and prove their basic properties for queries which only talk about properties of subtrees of a tree. In Section 4 we extend these results for queries of arbitrary arity and state corresponding results for the fragment of MSO. I'd like to thank Frank Neven, Wolfgang Thomas, Clemens Lautemann, Jan van den Bussche and the anonymous referees for helpful discussions and useful hints.

2 Preliminaries

Trees. In this paper, trees are rooted, unranked, directed graphs, where the children of a node are *ordered*. The vertices of a tree are labelled with labels from a fixed finite set Σ . Each vertex carries a set of labels from Σ . Edges are oriented from the root towards the leaves. We denote the *horizontal order* of the children of a vertex by $<$ and the *vertical order*, which is the transitive closure of the edge relation by $\preceq$. For a tree t and a vertex v of t we write t_v for the subtree of t which is rooted at v . For a tree t and vertices v, w with $v \preceq w$ we write $t(v, w)$ for the tree which results from t_v by deleting all vertices and edges below w and in which v and w are distinguished vertices.

Logic. A tree t over Σ can be naturally viewed as a finite structure over the binary relation symbols E (for the edges), $\preceq$ and $\leq$, the unary relation symbols

$(O_a)_{a \in \Sigma}$ (vertex labels), in the obvious way. We write $(t, v_1, \dots, v_m)$ to denote the tree t in which the vertices $v_1, \dots, v_m$ are distinguished.

First-order logic (FO) is defined as usual (see, e.g., [3]). For $n \geq 0$, a structure A and a tuple $\bar{v} = v_1, \dots, v_k$ of elements of A we write $\tau_n(A, \bar{v})$ to denote the set of all formulas $\varphi(\bar{x})$ of quantifier-depth at most n for which $A \models \varphi[\bar{v}]$. If A is clear from the context it might be omitted. We call $\tau_n(A, \bar{v})$ the n -type of $\bar{v}$ in A . The set of all possible n -types of formulas with k free variables is denoted by $\Phi_n(k)$. It is well-known (see again [3]) that, for each n and k , $\Phi_n(k)$ is finite.

In some of our proofs we are going to use a fragment of Monadic second-order logic (MSO) which is defined next. We assume two sets of set variables for horizontal and vertical quantification, respectively. The interpretation of vertical set variables is restricted to vertical chains, i.e., to sets of vertices that are completely ordered by $\preceq$. Analogously, horizontal set variables are only interpreted by sets that are completely ordered by $\leq$, i.e., to sets of children of a vertex. The resulting logic is called $\text{MSO}^{\text{chain}}$. This definition differs from [14], where, in the context of ranked trees, in $\text{MSO}^{\text{chain}}$ formulas only quantification of vertical chains is allowed. We denote types and sets of types wrt $\text{MSO}^{\text{chain}}$ by adding a superscript c to the corresponding FO notation. E.g., $\Phi_n^c(1)$ denotes the set of $\text{MSO}^{\text{chain}}$ types defined by $\text{MSO}^{\text{chain}}$ of quantifier depth n with one free variable. Types for full MSO logic will be indicated by a superscript 2.

We are especially interested in an extension of FO which allows the use of horizontal and vertical regular expressions. This extension was introduced in [9] as an attempt to capture the expressive power of existing query languages for semistructured data. It uses the following two kinds of path formulas.

- If P is a regular expression (in the usual sense) over formulas then $\varphi = [P]_{r,s}^\downarrow(x, y)$ is a *vertical path formula*. The free variables of φ are $\{x, y\} \cup (\text{free}(P) - \{r, s\})$, where $\text{free}(P)$ denotes the free variables that occur in at least one of the formulas that are used to build P .
- If P is a regular expression over formulas then $[P]_r^\rightarrow(x)$ is a *horizontal path formula*. The free variables of φ are $\{x\} \cup (\text{free}(P) - \{r\})$

We refer to path formulas also by the term *path expressions*. A simple example of a formula which uses a horizontal path expression is $[(O_a(r))^* O_b(r)]_r^\rightarrow(x)$.

The semantics of such formulas is defined as follows. Let $\varphi = [P]_{s,t}^\downarrow(x, y)$ be a vertical path formula, t a tree and v, w vertices of t . We assume interpretations for the free variables occurring in formulas in P . Then, $t \models \varphi[v, w]$, iff $v \preceq w$ and there is a labeling of the edges on the path from v to w with formulas, such that (1) each edge (u, u') is labeled with a formula $\theta(r, s)$ such that $t \models \theta[u, u']$, and (2) the sequence of labels along the path from v to w matches P .

For $\psi = [P]_r^\rightarrow(x)$, $t \models \psi[v]$, iff there is a labeling of the children of v with formulas, such that (1) each child w of v is labeled with a formula $\theta(r)$ such that $t \models \theta[w]$, and (2) the sequence of labels matches P . E.g., the above example formula says that the rightmost child of x is labelled with b and all other children are labelled with an a . The logic FOREG is obtained from FO by allowing vertical and horizontal path formulas. More formally, (1) Every FO

formula is an FOREG formula. (2) If P is a regular expression over FOREG formulas with free variables r, s then $\varphi = [P]_{r,s}^\downarrow(x, y)$ is an FOREG formula. (3) If P is a regular expression over FOREG formulas with free variable r then $[P]_r^\rightarrow(x)$ is an FOREG formula. This logic was called FOREG* in [9].

Games. Some of our proofs make use of Ehrenfeucht games. An n -round FO Ehrenfeucht game on two finite structures A and B is played by two players, often called the *spoiler* and the *duplicator*, respectively. In each round, the spoiler chooses a vertex in one of the two structures and the duplicator chooses afterwards a vertex in the other structure. Let the vertex that is chosen from A (B) in round i be a_i (b_i). The duplicator wins the game if the mapping which maps each a_i to b_i is a partial isomorphism. It is well known (see, e.g., [3]) that the spoiler has a winning strategy in the FO n -round game on A and B (denoted by $A \sim_n B$) if and only if there is an FO formula φ of quantifier rank $\leq n$ such that $A \models \varphi$ and $B \not\models \varphi$. In this case, A and B are indistinguishable wrt. formulas of quantifier rank up to n (written $A \equiv_n B$). Analogous game characterizations exist for MSO and $\text{MSO}^{\text{chain}}$ on trees. In the MSO game the spoiler can also choose sets of vertices, in the $\text{MSO}^{\text{chain}}$ game vertical or horizontal chains. The duplicator always has to respond with an object of the same kind. Following our general convention, we use the symbols $\sim_n^c$ and $\sim_n^2$ to denote the existence of a winning strategy for the duplicator in the $\text{MSO}^{\text{chain}}$ game and the MSO game, respectively.

3 GFO and GFOREG: Subtree-Restricted Queries

In this section we define and investigate fragments of FO and FOREG, respectively, which have the same expressive power on trees as FO and FOREG themselves but can be evaluated more efficiently. The definitions are in the spirit of the fragment of MSO logic on trees that was introduced in [9]. The main ingredients are guarded quantification and path formulas.

Intuitively, a subformula $\exists y\psi$ of an FO or FOREG formula is translated into a GFO or GFOREG formula of the kind $\exists y \bigvee_i (\psi'_i \wedge \psi''_i)$, where each ψ'_i is a vertical expression which talks about the tree above y and ψ''_i is a formula which only talks about the subtree rooted at y . In this section, we will define GFO and GFOREG formulas which can only express properties of subtrees that are rooted at one of their free variables. In section 4 we will add some more constructs which allow the expression of queries of arbitrary arity which involve the whole tree.

We use two kinds of FO-variables. One kind is called *quantifier variables* and used for quantification of vertices and is denoted by symbols like x, y . The second kind (*expression variables*, denoted by r, s) is only used in vertical or horizontal path expressions. The syntax of GFO formulas is defined as follows.

- (i) Every atomic formula is a GFO formula.
- (ii) If y is a quantifier variable and φ is a GFO formula with free quantifier variables from $\{x, y\}$ then $\exists y(x \preceq y \wedge \varphi)$ is a GFO formula.
- (iii) If P is a star-free regular expression over GFO formulas without free quantifier variables then $[P]_{r,s}^\downarrow(x, y)$ is a GFO formula.

- (iv) If P is a star-free regular expression over GFO formulas without free quantifier variables then $[P]_r^{\rightarrow}(x)$ is a GFO formula.
- (v) Any Boolean combination of GFO formulas is a GFO formula.

Let Σ be fixed for the rest of this section. GFOREG is defined in analogy to GFO but path expressions have to be regular expressions not star-free ones.

In a GFO formula $\varphi(x)$ quantification is restricted to vertices below x . To bridge the gap between FO and GFO we define a class of FO formulas that are less restricted than GFO. A formula φ is called *subtree-restricted wrt* a free variable x , if all quantifications in φ are of the form $\exists y(x \preccurlyeq y) \wedge \psi$. There are no restrictions on the free variables in ψ . If φ has only one free vertex variable we simply say that φ is subtree-restricted. We define $\tau_n^{\text{str}}(A, \bar{v})$ and $\Phi_n^{\text{str}}(k)$ in analogy to $\tau_n(A, \bar{v})$ and $\Phi_n(k)$, respectively, but consider only formulas that are subtree-restricted with respect to their first free variable.

First, we show that each subtree-restricted formula $\varphi(x)$ is equivalent to a GFO formula. To this end, we associate with each pair v, w of vertices in a tree t , for which $v \preccurlyeq w$ and each $n \geq 0$, a string $\text{vpath}_n(v, w)$ over the alphabet $\Phi_n(2)$ as follows. Let $u_1, \dots, u_m$ be the unique path from v to w , hence $u_1 = v, u_m = w$. Then $\text{vpath}_n(v, w) := a_1 \cdots a_{m-1}$ where, for each $i \leq m-1$, $a_i = \tau_n(t(u_i, u_{i+1}))$.

Lemma 1. *Let $n \geq 0$. For every FO formula $\varphi(x, y)$ of quantifier rank n which is subtree-restricted wrt x , there exists a finite set S of pairs (L, ψ) , where L is a star-free language over alphabet $\Phi_n^{\text{str}}(2)$, and $\psi(y)$ is a subtree-restricted formula of subtree-restricted quantifier-rank n , such that the following holds.*

For all trees t and vertices v and w of t with $v \preccurlyeq w$, $t \models \varphi[v, w]$ if and only if there is a pair (L, ψ) in S such that $\text{vpath}_n(v, w) \in L$ and $t \models \psi[w]$.

Proof. By using Ehrenfeucht games it is straightforward to prove the following.

Let t, t' be trees and let v, w be vertices from t and v', w' vertices from t' such that $v \preccurlyeq w$ and $v' \preccurlyeq w'$. If $\text{vpath}_n(v, w) \equiv_n \text{vpath}_n(v', w')$ and $(t_w, w) \equiv_n (t'_{w'}, w')$ then $(t_v, v, w) \equiv_n (t'_{v'}, v', w')$.

The proof of this claim is given in the full paper. We give a more detailed argument for a similar claim in the proof of Lemma 4 below.

From the claim we can conclude that whether $t \models \varphi[v, w]$ only depends on the n -types of $\text{vpath}_n(v, w)$ and (t_w, w) . From McNaughton's result that the first-order definable languages are exactly the star-free languages [6] it follows that for each n -type τ of $\Phi_n^{\text{str}}(2)$ -strings the set of all strings α with $\tau_n(\alpha) = \tau$ is star-free. Furthermore, each n -type of a subtree with one distinguished vertex can be characterized by a subtree-restricted formula of quantifier rank n . $\square$

With each vertex $w \neq \text{root}$ in a tree t , and each $n \geq 0$, we associate two strings $\text{hpathl}_n(w)$ and $\text{hpathr}_n(w)$ over the alphabet $\Phi_n^{\text{str}}(1)$ as follows. Let v be the parent of w and let $u_1, \dots, u_m$ be the children of v (in that order). Then $\text{hpathl}_n(w) := a_1 \cdots a_{j-1}$ and $\text{hpathr}_n(w) := a_{j+1} \cdots a_m$ where j is such that $u_j = w$ and, for each $i \leq m$, $a_i = \tau_n(t_{u_i})$.

Lemma 2. *For each formula $\varphi(x, y)$ of quantifier rank n which is subtree-restricted wrt x there is a finite set S of tuples $(L_1, L_2, \psi_1, \psi_2)$, where L_1, L_2 are star-free languages over $\Phi_n^{\text{str}}(1)$ and the ψ_1, ψ_2 are quantifier-free formulas with one free variable such that the following holds. For each tree t and all vertices v and w of t such that w is a child of v , it holds that $t \models \varphi[v, w]$ if and only if there is a tuple $(L_1, L_2, \psi_1, \psi_2)$ in S such that $\text{hpathl}_n(w) \in L_1$, $\text{hpathr}_n(w) \in L_2$, and ψ_1 and ψ_2 hold at v and w , respectively.*

The proof is quite similar to the proof of Lemma 1 above.

Theorem 3. *For every subtree-restricted FO formula $\varphi(x)$ there is a GFO formula $\psi(x)$ such that, for all trees t and vertices v from t it holds*

$$t \models \varphi[v] \iff t \models \psi[v].$$

Proof. The proof is by induction on the quantifier rank of φ . If it is 0 the result follows directly as each quantifier free formula is a GFO formula.

Now assume that the quantifier rank of φ is $n+1$, for some $n \geq 0$. As GFO is closed under Boolean combinations it is sufficient to consider formulas φ of the form $\exists y(x \preceq y \wedge \theta(x, y))$, where θ is subtree-restricted wrt x and has quantifier rank n . By Lemma 1 there is a finite set S of pairs (L, ψ) , where L is a star-free language over $\Phi_n^{\text{str}}(2)$, and $\psi(y)$ is a subtree-restricted formula of quantifier-rank n , such that for each tree t and vertices v and w of t , with $v \preceq w$, $t \models \theta[v, w]$ if and only if there is a pair (L, ψ) in S such that $\text{vpath}_n(v, w) \in L$ and $t \models \psi[w]$.

Now for each $\tau \in \Phi_n^{\text{str}}(2)$ there is a subtree-restricted FO formula $\chi(x, y)$ of quantifier rank n such that $t(v, w)$ is of type τ if and only if $t \models \chi[v, w]$. By Lemma 2 there exists for each such χ a finite set S_χ of tuples $(L_1, L_2, \psi_1, \psi_2)$ such that for each tree t and all vertices v and w of t such that w is a child of v it holds that $t \models \chi[v, w]$ if and only if there is a tuple $(L_1, L_2, \psi_1, \psi_2)$ in S_χ such that $\text{hpathl}_n(w) \in L_1$, $\text{hpathr}_n(w) \in L_2$, and ψ_1 and ψ_2 hold for the single vertex tree consisting of v and w , respectively. As χ will only be used in a vertical expression, we can choose its free variables as expression variables r, s .

Hence, $t \models \chi[v, w]$ if and only if, for some $(L_1, L_2, \psi_1, \psi_2)$ in S_χ , $(t, v, w) \models [P_1 \cdot ((r' = s) \wedge \psi_2(r)) \cdot P_2]_{r', s}^{\downarrow}(r, s)$ and, at the same time, $t \models \psi_1[v]$. Here P_1 and P_2 are star-free expressions over subtree-restricted formulas of quantifier rank n , which describe L_1 and L_2 , respectively. By induction, the formulas in P_1 and P_2 can be replaced by equivalent GFO formulas. By taking the disjunction over S_χ we get a GFO formula which is equivalent to χ . Hence, for each (L, ψ) from S there is a star-free expression P over GFO formulas such that $\text{vpath}_n(v, w) \in L$ if and only if $t \models [P]_{r, s}^{\downarrow}[v, w]$. Combining this with ψ and taking the disjunction over all pairs in S we get a GFO formula θ' which is equivalent to θ . Hence, $\exists y(x \prec y \wedge \theta')$ is a GFO formula equivalent to φ . $\square$

It has been already stated in [9] that for every FOREG formula there is an equivalent $\text{MSO}^{\text{chain}}$ formula. As a byproduct of the following development it will turn out that these two logics have actually the same expressive power. Before we show that GFOREG can express all subtree-restricted FOREG queries we first prove two decomposition lemmas for $\text{MSO}^{\text{chain}}$.

The notion of *subtree-restricted* formulas is generalized to $\text{MSO}^{\text{chain}}$ and MSO formulas by restricting the range of the quantified set with respect to the variable in question (denoted, e.g., as $x \preccurlyeq X$). Following our notation policy, in the following, $\text{vpath}_n^c(v, w)$ is the analogue of $\text{vpath}_n^c(v, w)$ over $\Phi_n^c(2)$ formulas.

Lemma 4. *For each subtree-restricted $\text{MSO}^{\text{chain}}$ formula $\varphi(x, X)$ of quantifier rank n with a vertical set variable X there is a regular language L over $\Phi_n^{\text{c, str}}(2)$ such that the following holds.*

For each tree t and vertex v of t there exists a vertical chain C in t_v with $t \models \varphi[v, C]$ if and only if there is a leaf w in t_v such that $\text{vpath}_n^c(v, w) \in L$.

Proof. We first prove the following claim.

Let t, t' be trees with vertices v, w and v', w' , respectively, such that $v \preccurlyeq w$ and $v' \preccurlyeq w'$. If $\text{vpath}_n^c(v, w) \equiv_{2n+1}^c \text{vpath}_n^c(v', w')$ then for each vertical chain C which is a subset of the path from v to w there is a vertical chain C' which is a subset of the path from v' to w' such that $(t_v, v, C) \equiv_n^c (t'_{v'}, v', C')$.

We show that C' can be chosen such that the duplicator has a winning strategy in the n -round $\text{MSO}^{\text{chain}}$ -game on (t_v, v, C) and $(t'_{v'}, v', C')$. Assume that k rounds of the game have been played. We define some notation for vertices and chains in t (and the corresponding for t').

For notational convenience, we define for every move i a vertical chain C_i , a horizontal chain D_i and a vertex u_i . We set $C_i = \emptyset$ if the i -th move was not a vertical chain move otherwise it is the selected chain. Analogously for D_i . We set $u_i = v$ if the i -th move was not a vertex move otherwise it is the selected vertex. Further, we associate with each move i a vertex v_i on the path p from v to w as follows. If the i -th move was a vertex move then v_i is the lowest vertex of p such that u_i is in the subtree rooted at v_i . If it was a vertical chain move then v_i is the lowest vertex on p which is compatible with C_i , i.e., such that $C_i \cup \{v_i\}$ is still a vertical chain. If it was a horizontal chain move then v_i is the lowest vertex of p such that the parent of the vertices in D_i is in the subtree rooted at v_i . We write $\text{child}(v_i)$ for the child of v_i on p (and set $\text{child}(v_i) = w$, if $v_i = w$ - in this case $t(v_i, \text{child}(v_i))$ shall denote t_{v_i}).

We also associate with each move i a set E_i on p . If it is a vertex move then $E_i = \emptyset$. If it is a vertical or horizontal chain move then E_i is the set of vertices from p that are in the chain. For each $i, j \leq k$ we define vertices u_i^j , vertical chains C_i^j and horizontal chains D_i^j as follows. If u_i is in $t(v_j, \text{child}(v_j))$ then $u_i^j = u_i$ otherwise it is v_j . The chain C_i^j is defined as the intersection of C_i with $t(v_j)$ and, correspondingly, D_i^j is the intersection of D_i with $t(v_j)$. Note that, for each i and j , only one of u_i^j, C_i^j, D_i^j is nontrivial.

It can be shown by induction on k that the duplicator can play in a way that assures that after k rounds the following conditions hold.

- (a) $(\text{vpath}_n^c(v, w), E_1, \dots, E_k, v_1, \dots, v_k) \sim_{2(n-k)}^c (\text{vpath}_n^c(v', w'), E'_1, \dots, E'_k, v'_1, \dots, v'_k)$.

(b) For each $j \leq k$,

$$(t(v_j, \text{child}(v_j)), C_1^j, \dots, C_k^j, D_1^j, \dots, D_k^j, u_1^j, \dots, u_k^j) \sim_{n-k}^c (t'(v'_j, \text{child}(v'_j)), C_1'^j, \dots, C_k'^j, D_1'^j, \dots, D_k'^j, u_1'^j, \dots, u_k'^j).$$

For $k = 0$, (a) and (b) follow directly from the preconditions of the claim. The inductive step makes use of the fact, that each move of the spoiler in the game on, say, t_v induces one vertex and one set on $\text{vpath}_n^c(v, w)$ and at most one non-trivial object (horizontal chain, vertical chain or vertex) in each of the subtrees $t(v_j, \text{child}(v_j))$. Furthermore, a chain move induces a nontrivial chain for at most one subtree $t(v_j, \text{child}(v_j))$. Hence, the winning strategies of duplicator that are inductively guaranteed on the games from (a) and (b) induce a response for the duplicator in the global game, which in turn maintains (a) and (b).

For $k = n$, (a) and (b) imply the above claim. Hence, whether a vertical chain C with $t \models \varphi[v, C]$ exists depends only on the set of $(2n + 1)$ -types of $\text{vpath}_n^c(v, w)$. Therefore, we get the desired set L by taking the union of all those regular languages over $\Phi_n^c(2)$ which imply the existence of such a C . $\square$

Lemma 5. *For each subtree-restricted $\text{MSO}^{\text{chain}}$ formula $\varphi(x, X)$ of quantifier rank n with a horizontal set variable X there is a set S of triples (L, ψ, L') where each L and L' are regular languages over $\Phi_n^{c, \text{str}}(1)$ and $\psi(x)$ is a quantifier-free formula such that the following holds.*

For each tree t and vertex w of t_v there is a horizontal set C of children of w with $t \models \varphi[v, C]$ if and only if there is a triple (L, ψ, L') in S such that $\text{vpath}_n^c(v, w) \in L$, $\text{hpath}_n^c(w) \in L'$ and ψ holds at w .

The proof is similar to the proof of Lemma 4.

Theorem 6. *For every subtree-restricted FOREG formula $\varphi(x)$ there is a GFOREG formula $\psi(x)$ such that, for all trees t and vertices v from t it holds*

$$t \models \varphi[v] \iff t \models \psi[v].$$

Proof. Again, we can only give a sketch of the proof. It is sufficient to show that for each $\text{MSO}^{\text{chain}}$ formula φ there exists a GFOREG formula ψ with the stated property. The proof of this statement is very similar to the proof of Theorem 3. In particular, it proceeds by induction on the quantifier rank n of φ . We give a hint on the argument for formulas of the kind $\exists X (x \preceq X \wedge \theta)$ with a vertical X . By Lemma 5 there exists a regular language L over $\Phi_{n-1}^{c, \text{str}}$ such that there is a vertical chain C with $t \models \theta[C, v]$ if and only if $\text{hpath}_n(w) \in L$, for some leaf $w \in t_v$. Hence, φ is equivalent to

$$\exists y (x \preceq y \wedge (\neg \exists z (y \preceq z \wedge y \neq z))) \wedge [P]_{r,s}^\downarrow(x, y),$$

where P is a regular expression over subtree-restricted FOREG formulas of quantifier depth $n - 1$. In a way similar to the proof of Theorem 3, we can conclude that the formulas in P can be replaced by equivalent GFOREG formulas. $\square$

Now we turn to query evaluation. From the result in [9] on the similarly defined fragment of MSO logic we immediately get the following.

Proposition 7. *There is an algorithm which computes on input (t, φ) , where t is a tree and φ is a GFOREG formula, the set of all vertices v of t such that $t \models \varphi[v]$ in time $O(|t|2^{|\varphi|})$.*

4 GFO and GFOREG: Arbitrary Arity

In this section, it will turn out that MSO, FOREG and FO queries of arbitrary arity can be expressed by suitable combinations of subtree-restricted unary (MSO, FOREG and FO, respectively) queries by means of regular path expressions, an operator lca which computes the least common ancestor of two vertices and an additional horizontal path operator. Due to lack of space we omit the proof and the statement of the underlying decomposition lemma.

We write $v \leq w$ if $v \preceq w$ or the subtree of $\text{lca}(v, w)$ containing v is left of the subtree containing w . If $v_i \leq v_j$, we write $\text{hpathm}_n^2(v_i, v_j)$ for the string $a_1 \cdots a_m$ which is obtained as follows. Let u and u' be the children of $\text{lca}(v_i, v_j)$ that contain v_i and v_j in their subtrees, respectively, and let $u_1, \dots, u_m$ be the children of $\text{lca}(v_i, v_j)$ that are located between u and u' . Then, for each i , a_i is defined as the MSO n -type of u_i .

Let S be a set of element variables. We call a term over S and the symbol root which uses the binary function symbol lca an *lca-term over S* . If P is a regular expression over formulas of quantifier rank n then $[P]_r^m(x, y)$ is an *intermediate horizontal path formula*. Such a formula holds true for vertices v, w of a tree t , if $v \leq w$ and $\text{hpathm}_n(v, w)$ matches P . Let $x_1, \dots, x_k$ be some variables. Let $\mathcal{L}$ be one of the logics FO, FOREG, $\text{MSO}^{\text{chain}}$, MSO. A *modular $\mathcal{L}$ -expression* over $x_1, \dots, x_k$ is a Boolean combination of subtree-restricted $\mathcal{L}$ -formulas $\varphi(f)$ where f is an lca-term over $x_1, \dots, x_k$, and formulas $[P]_{r,s}^k(f, g)$, $[P]_{\rightarrow}^k(f)$ and $[P]_r^m(f, g)$ where f and g are lca-terms and P is a regular expression over $\mathcal{L}$ -formulas.

If $\mathcal{L}$ is FO then the path expressions have to be starfree.

Proposition 8. *Let $\mathcal{L}$ be one of the logics FO, FOREG, $\text{MSO}^{\text{chain}}$, MSO. For each $\mathcal{L}$ -formula $\varphi(\bar{x})$ there is a modular $\mathcal{L}$ -expression $\psi(\bar{x})$ such that for all trees t and tuples $\bar{v}$ of vertices it holds*

$$t \models \varphi[\bar{v}] \iff t \models \psi[\bar{v}].$$

If $\mathcal{L}$ is one of FO, FOREG and MSO, a *modular $\mathcal{GL}$ -expression* is defined like a modular $\mathcal{L}$ -expression but the unary subformulas are restricted to GFO (GFOREG or the fragment of MSO from [9], resp.).

In Section 3 it was shown that unary subtree-restricted FO and FOREG formulas can be replaced by GFO and GFOREG formulas, respectively. For the analogue restriction of MSO the corresponding result was shown in [9]. By combining these results with Proposition 8 we obtain the following theorem.

Theorem 9. Let $\mathcal{L}$ be one of the logics FO, FOREG, MSO. For each $\mathcal{L}$ -formula $\varphi(\bar{x})$ there is a modular $\mathcal{GL}$ -expression $\psi(\bar{x})$ such that for all trees t and tuples $\bar{v}$ of vertices it holds

$$t \models \varphi[\bar{v}] \iff t \models \psi[\bar{v}].$$

As a byproduct we get the following corollary.

Corollary 10. On trees MSO^{chain} and FOREG can express the same queries.

Concerning evaluation complexity we get the following.

Theorem 11. (a) There is an algorithm which checks in time $O(|t|2^{|\varphi|})$ on input $(t, \bar{v}, \varphi)$, where t is a tree, $\bar{v}$ is a tuple of vertices of t and φ is a modular GMSO or GFOREG expression, whether $t \models \varphi[\bar{v}]$.
 (b) There is an algorithm which computes in time $O(|t|2^{|\varphi|})$ on input (t, φ) , where t is a tree and φ is a GFOREG formula, a data structure which allows to check in time $O(|\varphi|)$, whether $t \models \varphi[\bar{v}]$ holds, for a given tuple $\bar{v}$.

References

1. Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.
2. J. Doner. Tree acceptors and some of their applications. *Journal of Computer and System Sciences*, 4:406–451, 1970.
3. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1995.
4. M. Fernandez, J. Siméon, and P. Wadler. XML query languages: Experiences and exemplars. <http://www-db.research.bell-labs.com/user/simeon/xquery.html>, 1999.
5. T. Hafer and W. Thomas. Computation tree logic CTL and path quantifiers in the monadic theory of the binary tree. In *ICALP*, pages 269–279, 1987.
6. R. McNaughton and S. Papert. *Counter-Free Automata*. MIT Press, 1971.
7. A. Neumann and H. Seidl. Locating matches of tree patterns in forests. In V. Arvind and R. Ramanujam, editors, *FST & TCS*, LNCS, pages 134–145. Springer, 1998.
8. F. Neven and T. Schwentick. Query automata. In *PODS*, pages 205–214. ACM Press, 1999.
9. F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. *PODS 2000*, 2000.
10. F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. In *PODS*, pages 11–17. ACM Press, 1998.
11. L. Stockmeyer. The complexity of decision problems in automata and logic, 1974. Ph.D. Thesis, MIT, 1974.
12. J.W. Thatcher and J.B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Mathematical Systems Theory*, 2(1):57–81, 1968.
13. W. Thomas. Logical aspects in the study of tree languages. In B. Courcelle, editor, *Proceedings of the 9th International Colloquium Trees in Algebra and Programming*, pages 31–50. Cambridge University Press, 1984.
14. W. Thomas. On chain logic, path logic, and first-order logic over infinite trees. In *LICS*, pages 245–256, 1987.

Abstract Syntax and Variable Binding for Linear Binders

Miki Tanaka

Graduate School of Informatics, Kyoto University, Japan
tmiki@kuis.kyoto-u.ac.jp Fax: +81-75-753-4954

Abstract. We apply the theory of binding algebra to syntax with linear binders. We construct a category of models for a linear binding signature. The initial model serves as abstract syntax for the signature. Moreover it contains structure for modelling simultaneous substitution. We use the presheaf category on the free strict symmetric monoidal category on 1 to construct models of each binding signature. This presheaf category has two monoidal structures, one of which models pairing of terms and the other simultaneous substitution.

1 Introduction

From the perspective of semantics, the essential syntactical entity of programming languages that we are mostly concerned with is not concrete syntax, like the one given by BNF grammars, but rather a structure that is more abstract. The distinction becomes apparent when one considers *variable binding*; in concrete syntax, it is realised by introducing a binder that specifies a variable and a scope so that the argument for that scope should be substituted for the specified variable. But, the name for the bound variable is irrelevant for what we want to do. For instance, take two lambda terms, $\lambda x.x$ and $\lambda y.y$. Obviously they are distinct terms; but both denote the identity function. What we need here is only position in the term where the variable that the argument is to be substituted for should appear: we are interested in terms modulo renaming of bound variables, or α -equivalence classes of terms.

One way to address this is to consider each concrete term as a specific representative of the α -equivalence class to which the term belongs and to allow the replacement of a term by any α -equivalent term. De Bruijn showed two systematic ways to choose the representatives consistently [4]. There have been various efforts to establish a syntax that allows one to deal with α -equivalence classes of terms directly. This kind of syntax is sometimes called *abstract syntax* [14].

Recently, Fiore et al. and independently, Gabbay and Pitts, developed a categorical algebraic theory of abstract syntax for signatures with variable binding in [6] and [8] respectively. In this paper, we consider a categorical algebraic description of syntax with *linear* variable binding. We apply the construction in [6] to signatures with linear binders, as appear for instance in the linear lambda calculus, the implicational fragment of the term calculus of linear logic [3,10].

Linear binders also appear in linear versions of other calculi, such as pi-calculus and action calculus [1]. We use the category P , the free strict symmetric monoidal category on 1 instead of F in [6], the free category with finite products on 1.

In dealing with substitution, the construction for single-variable substitution in [6], which worked for non-linear binders, cannot be applied to linear binders, while the construction for simultaneous substitution extends readily to linear binders. See the end of Section 4 for detail. Our construction for linear binders coincides with part of Joyal’s work on species [12,13,2]; the functor $\hat{\otimes}$ for the pairing of linear terms is equivalent to the partitional product of species, while the functor $\bullet$ for simultaneous substitution of linear terms is equivalent to the substitution of species. This correspondence is consistent with the combinatorial nature of linear binding.

We have many possibilities for future work. In this paper, we give a theory to deal with linear binders while Fiore et al. gave an account for the usual (non-linear) binders in [6]. These are both for untyped languages, so naturally, to extend the theory of binding algebra to typed languages, such as simply typed lambda calculus, simply typed linear lambda calculus, and FPC [7], is one direction for future work. Some call-by-value binders, such as binders for the computational lambda calculus [15] will also be investigated. Moreover, incorporation of the various above mentioned binders into the axiomatics for structural operational semantics should be explored [21]. And we should also seek a unified framework for binding algebras for usual binders in [6] and for linear binders in this paper, and the combinations of these for other language features. A generalised study of algebraic structure on categories would be a way to address that problem.

Linear variables and linear binding have much significance in their own right. In a functional language, variables can be copied or discarded at will. But in an imperative language, it makes little sense in general to attempt to copy or discard the whole state. So state is inherently linear, whereas variables for functions are not. This linearity of state has long been implicitly discussed, albeit without using the term *linear*, since 70’s. As Strachey wrote, “The state transformation produced by obeying a command is essentially irreversible and it is, by the nature of the computers we use, impossible to have more than one version of [the state] available at any one time,” in [19]. More recently, O’Hearn and Reynolds have given an account of this irreversibility in procedural languages in [16], by giving translations into polymorphic linear lambda calculus. In doing so, they explicitly used two different binders, one for ordinary binding and the other for linear binding. Therefore topics related to linear binding are not just abstract concerns but have direct applications, for instance, in interpreting languages with both functional and imperative features, such as idealisations of Algol [17,18]: there one needs a delicate interaction of linear and non-linear substitution.

The paper is organised as follows; In Section 2, we construct algebraic structure for linear binders. With the leading example of the linear lambda calculus, we introduce the category of linear contexts $\mathbb{P}$ and linear binding signatures.

Then we construct algebras for linear binding signatures. In Section 3, we give a construction for simultaneous substitution by using a monoidal structure on the presheaf category $[\mathbb{P}, \text{Set}]$. In Section 4, we define the category of models for a linear binding signature, by merging the algebra for the signature and the construction for simultaneous substitution. The presheaf corresponding to the terms of the signature is the initial model.

2 Linear Binding Algebra

In this section, we give the definition of linear binding signatures and construct the category of their algebras. But first, we introduce the notion of abstract linear context, with the linear lambda calculus as the leading example, and then give the description of the category with which we shall work.

2.1 Linear Contexts

What we call linear binding here is variable binding with the following two properties:

- No variable can occur free in a term more than once.
- A binder cannot bind a variable that does not appear in the term.

These conditions can be realised in a uniform manner by imposing well-formedness rules with special contexts on the usual (non-linear) terms.

Definition 2.1. *A linear context is a sequence of variables, with no variables occurring twice.*

This definition coincides with that for usual contexts, but operations on linear ones differ from those on usual ones, defined as follows: given linear contexts Γ and Δ with no common variables, a *merge* of Γ and Δ is the linear context which is a sequence of all variables in Γ and Δ appearing in an arbitrary order. So we have $(m+n)!$ possible merges of Γ and Δ which contain m and n variables respectively. We write $\Gamma \# \Delta$ to stand for any context obtained by merging Γ and Δ [3,10]. On the other hand, Γ, Δ means the usual concatenation of contexts. The following example shows how linear contexts are used to form linear terms.

Example 2.1. The terms of untyped linear lambda calculus are obtained by imposing the above two conditions on (non-linear) untyped lambda terms.

$$x \vdash_1 x \qquad \frac{\Gamma, x \vdash_1 t}{\Gamma \vdash_1 \lambda x. t} \qquad \frac{\Gamma \vdash_1 t_1 \quad \Delta \vdash_1 t_2}{\Gamma \# \Delta \vdash_1 t_1 t_2}$$

where Γ and Δ are linear contexts.

Observe that we have no contraction or weakening of contexts; we only have exchange of contexts. And another point to note is that free variables in a term must be controlled rigidly by the context, rather than be included in it. In other words, every variable in the context must appear in the terms derived from that context.

Proposition 2.1. *Let t be a linear lambda term. If $x_1, \dots, x_n \vdash_1 t$ then $FV(t)$, the set of free variables of t , is equal to the set $\{x_1, \dots, x_n\}$.*

Based on this proposition, we go further to take up a more abstract view of linear contexts. In the following, we write n for the linear context $\{x_1, \dots, x_n\}$ and call it an *abstract linear context*. The merge of abstract linear contexts is just the sum of natural numbers, and the permutation of indices, which is part of the definition of the merge of linear contexts, is passed on to the operation of pairing terms. So Example 2.1 may be reformulated as follows;

Example 2.2. The rules below give the canonical representative for α -equivalence classes of linear lambda terms by the method of de Bruijn levels [4];

$$1 \vdash_2 x_1 \quad \frac{n+1 \vdash_2 t}{n \vdash_2 \lambda x_{n+1}.t} \quad \frac{n \vdash_2 t_1 \quad m \vdash_2 t_2}{n+m \vdash_2 t'_1 t'_2}$$

where $t'_1 t'_2$ is a term obtained from $t_1 t_2$ by first shifting the indices of variables in t_2 by n , and then permuting the indices of variables in both t_1 and t_2 by an arbitrary permutation of $m+n$.

We take as the category of abstract linear contexts the category $\mathbb{P}$ with objects $n = \{0, \dots, n-1\}$ ($n \in \mathbb{N}$) and $\mathbb{P}(n, m) = \emptyset$ for all $n, m \in \mathbb{N}$ such that $n \neq m$ and $\mathbb{P}(n, n) = S_n$ for all $n \in \mathbb{N}$, where S_n is the set of permutations on an n -element set. This $\mathbb{P}$ is isomorphic to the free strict symmetric monoidal category on 1. The tensor product for the monoidal structure is given by sum of natural numbers, but to avoid confusion, we denote it by $\otimes$. We call the functor $-\otimes 1 : \mathbb{P} \rightarrow \mathbb{P}$ *context extension*, which sends n to $n+1$ for all $n \in \mathbb{P}$. And for each $\sigma \in S_n$, $\sigma \otimes 1 : n+1 \rightarrow n+1$ is

$$(\sigma \otimes 1)(i) = \begin{cases} \sigma(i) & \text{if } 0 \leq i \leq n-1 \\ n & \text{if } i = n \end{cases}$$

Abstract linear contexts stratify terms, just as abstract cartesian contexts do in [6].

Example 2.3. Define $\Lambda_\alpha, \Lambda_l : \mathbb{P} \rightarrow \text{Set}$ as follows: for all $n \in \mathbb{N}$,

$$\begin{aligned} \Lambda_\alpha(n) &= \{[t]_\alpha \mid x_1, \dots, x_n \vdash_1 t\} \\ \Lambda_l(n) &= \{t \mid n \vdash_2 t\} \end{aligned}$$

where $[t]_\alpha$ denotes the α -equivalence class of t . Clearly we have $\Lambda_\alpha(n) \cong \Lambda_l(n)$, by the assignment $[t]_\alpha \mapsto t$.

The above example justifies the use of the presheaf category $[\mathbb{P}, \text{Set}]$ as the domain in which we develop our discussion.

2.2 Constructions on $[\mathbb{P}, \text{Set}]$

In this subsection, we give several definitions of operations on $[\mathbb{P}, \text{Set}]$ that we need to model signatures with linear binding.

Let $\mathcal{Y} : \mathbb{P}^{\text{op}} \rightarrow [\mathbb{P}, \text{Set}]$ be the Yoneda functor. We call $\mathcal{Y}_1 = \mathbb{P}(1, -)$, the Yoneda embedding of 1 into $[\mathbb{P}, \text{Set}]$, the presheaf of *abstract linear variables*.

Definition 2.2. $\partial : [\mathbb{P}, \text{Set}] \rightarrow [\mathbb{P}, \text{Set}]$ is the functor obtained by precomposition by $- \otimes 1$, sending $F : \mathbb{P} \rightarrow \text{Set}$ to $F \circ (- \otimes 1) : \mathbb{P} \rightarrow \text{Set}$.

The functor ∂ is used to model linear binding of a variable.

In [6], finite products in the category $[\mathbb{P}, \text{Set}]$ were used to model pairing of terms. Our category $[\mathbb{P}, \text{Set}]$ also has finite products, but in our case, finite products do not model pairing adequately, as the linearity condition requires that the free variables in terms should be controlled rigidly by the contexts. Instead we use another symmetric monoidal structure on $[\mathbb{P}, \text{Set}]$, which is known as Day's tensor, given as follows [5]:

Definition 2.3. A bifunctor $\widehat{\otimes} : [\mathbb{P}, \text{Set}] \times [\mathbb{P}, \text{Set}] \rightarrow [\mathbb{P}, \text{Set}]$ is obtained as the left Kan extension of $\mathcal{Y} \circ (- \otimes -)$ along $\mathcal{Y} \times \mathcal{Y}$. For $F, G \in [\mathbb{P}, \text{Set}]$ and $n \in \mathbb{P}$, $F \widehat{\otimes} G(n)$ is defined as

$$F \widehat{\otimes} G(n) = (\coprod_{n=m_1+m_2} Fm_1 \times Gm_2 \times S_n) / \sim$$

where the equivalence relation is generated by the relation

$$(x, y, \sigma \circ (\sigma_1 \otimes \sigma_2)) \sim ((F\sigma_1)x, (G\sigma_2)y, \sigma). \quad (1)$$

Some simple calculations show that $\mathcal{Y}_0 = \mathbb{P}(0, -)$ is both left and right identity for $\widehat{\otimes}$. The symmetry isomorphism $c : F \widehat{\otimes} G \xrightarrow{\sim} G \widehat{\otimes} F$ is given by

$$\begin{aligned} c_n : (F \widehat{\otimes} G)n &\xrightarrow{\sim} (G \widehat{\otimes} F)n \\ [(x, y, \sigma)] &\mapsto [(y, x, \sigma \circ \theta)] \end{aligned}$$

where θ is defined as follows: for $x \in Fm_1, y \in Gm_2$, and $n = m_1 + m_2$,

$$\theta(i) = \begin{cases} i + m_2 & \text{if } 0 \leq i \leq m_1 - 1 \\ i - m_1 & \text{if } m_1 \leq i \leq n - 1. \end{cases}$$

Moreover, for any $X \in [\mathbb{P}, \text{Set}]$, $-\widehat{\otimes} X$ has a right adjoint $X \multimap -$ where $(X \multimap G)(n) = [\mathbb{P}, \text{Set}](X, G \circ (- \otimes n))$ for $G \in [\mathbb{P}, \text{Set}]$ and $n \in \mathbb{N}$. It follows that $[\mathbb{P}, \text{Set}]$ has a symmetric monoidal closed structure with this tensor $\widehat{\otimes}$ [5,11].

Using the above adjunction, by setting $X = \mathcal{Y}_1$, we have $-\widehat{\otimes} \mathcal{Y}_1 \dashv \partial$ or equivalently $\mathcal{Y}_1 \multimap F \cong \partial F$ for any $F \in [\mathbb{P}, \text{Set}]$. In fact, ∂ also has a right adjoint given by right Kan extension.

We denote $G \widehat{\otimes} \cdots \widehat{\otimes} G$, the k -fold tensor product of G , by $(\widehat{\otimes} G)^k$. We can construct the reordering map $R_\rho : G_1 \widehat{\otimes} \cdots \widehat{\otimes} G_k \xrightarrow{\sim} G_{\rho 1} \widehat{\otimes} \cdots \widehat{\otimes} G_{\rho k}$ for each permutation $\rho : k \xrightarrow{\sim} k$, by repeated use of instances of the symmetry c .

$$R_{\rho, n} : [(u_1, \dots, u_k, \sigma)] \mapsto [(u_{\rho 1}, \dots, u_{\rho k}, \sigma')] \quad (2)$$

where $(u_1, \dots, u_k) \in G_1 m_1 \times \cdots \times G_k m_k$, $(u_{\rho 1}, \dots, u_{\rho k}) \in G_{\rho 1} m_{\rho 1} \times \cdots \times G_{\rho k} m_{\rho k}$, $n = m_1 + \cdots + m_k$, and $\sigma, \sigma' \in S_n$. The permutation σ' is obtained from σ by applying suitable compositions of the symmetry c .

Note that an equivalence class $[(x, y, \sigma)]$ of the relation in (1) is in effect a pair of terms (x, y) coupled with a partition of $\{0, \dots, n-1\}$ into U_1 and U_2 ,

such that $|U_1| = m_1$, $x \in Fm_1$ and $|U_2| = m_2$, $y \in Gm_2$. This shows that the redundancy mentioned in Example 2.1 is resolved here. Moreover, one can immediately see that the tensor $\widehat{\otimes}$ is equivalent to the multiplication operation of species [2,12,13].

2.3 Linear Binding Signatures and Their Algebras

Here we define linear binding signatures, associate with each signature an endofunctor on $[\mathbb{P}, \text{Set}]$, and construct an algebra structure for the signature.

Definition 2.4. A binding signature $\Sigma = (O, a)$ is a pair of a set of operations O and an arity function $a : O \rightarrow \mathbb{N}^*$ [6]. An operator of arity $\langle n_1, \dots, n_k \rangle$ has k arguments and binds n_i variables in the i -th argument ($1 \leq i \leq k$). Here we consider only linear binders, so each variable bound by a binder has exactly one occurrence in the term. In the following, we call Σ a linear binding signature.

Example 2.4. The signature for linear lambda calculus $\Sigma_{\lambda l}$ has two operators, abstraction λ and application $@$, of arity $\langle 1 \rangle$ and $\langle 0, 0 \rangle$ respectively.

The terms t associated with a linear binding signature Σ over a set of variables are given by the following grammar, which is the same as the one for usual non-linear binding signatures [6]:

$$t \in T_{\Sigma} ::= x \mid o((x_1, \dots, x_{n_1}).t_1, \dots, (x'_1, \dots, x'_{n_k}).t_k)$$

where o is an operator of arity $\langle n_1, \dots, n_k \rangle$. The definitions of free and bound variables and α -equivalence are obtained in the obvious way. For each linear binding signature, we have a presheaf of terms modulo α -equivalence $\text{TV}_{\alpha} \in [\mathbb{P}, \text{Set}]$, with $\text{TV}_{\alpha}(n) = \{[t]_{\alpha} \mid FV(t) = \{x_1, \dots, x_n\}\}$, just as in Example 2.3.

We associate a functor $\Sigma : [\mathbb{P}, \text{Set}] \rightarrow [\mathbb{P}, \text{Set}]$ to a linear binding signature $\Sigma = (O, a)$.

$$\Sigma X = \coprod_{\substack{o \in O \\ a(o) = \langle n_i \rangle_{1 \leq i \leq k}}} (\partial^{n_1} X) \widehat{\otimes} (\partial^{n_2} X) \widehat{\otimes} \dots \widehat{\otimes} (\partial^{n_k} X).$$

Binding one variable linearly corresponds to the functor ∂ and pairing two terms corresponds to the functor $\widehat{\otimes}$. From the functor Σ we form the category $\Sigma\text{-Alg}$ of Σ -algebras, whose objects are *algebras* (A, h) , where $h : \Sigma A \rightarrow A$ is a morphism in $[\mathbb{P}, \text{Set}]$, and whose morphisms $f : (A, h) \rightarrow (A', h')$ are morphisms $f : A \rightarrow A'$ in $[\mathbb{P}, \text{Set}]$ such that $f \circ h = h' \circ \Sigma f$.

Let U be the forgetful functor from $\Sigma\text{-Alg}$ to $[\mathbb{P}, \text{Set}]$, with $U(A, h) = A$. This U has a right adjoint that carries each presheaf X to TX , the free Σ -algebra over X . This TX is computed as $TX = \coprod_{n \in \mathbb{N}} (X + \Sigma)^n(\emptyset)$, with X regarded as a constant functor on $[\mathbb{P}, \text{Set}]$, and with $\emptyset \in [\mathbb{P}, \text{Set}]$ being the presheaf which sends all n to $\emptyset$. Since $X + \Sigma$ preserves colimits, we have $\Sigma TX \cong TX$ and so we can take the canonical isomorphism as the morphism for the algebra.

Theorem 2.1. *TV_α associated to a linear binding signature is a free Σ -algebra on the presheaf of linear variables $\mathcal{Y}_1$.*

Example 2.5. For the case of the signature of linear lambda calculus $\Sigma_{\lambda l}(X) = \partial X + X \hat{\otimes} X$, the calculation of the free algebra Λ on the presheaf of variables $\mathcal{Y}_1$ corresponds to the following inductive definitions; $\Lambda(n) = \{t \mid n \vdash t\}$ ($n \in \mathbb{P}$) where

$$1 \vdash \text{var}(1) \quad \frac{n+1 \vdash t}{n \vdash \text{lam}(t)} \quad \frac{n \vdash t_1 \quad m \vdash t_2}{n+m \vdash \text{app}([t_1, t_2, \sigma])}$$

and for $\tau : n \xrightarrow{\sim} n$ and $t \in \Lambda(n)$,

$$\begin{aligned} \Lambda(\tau)(t) = \text{case } t \text{ of} \\ \text{var}(1) &\Rightarrow \text{var}(1) \\ \text{lam}(t') &\Rightarrow \text{lam}(\Lambda(\tau \otimes 1)(t')) \\ \text{app}([t_1, t_2, \sigma]) &\Rightarrow \text{app}([t_1, t_2, \tau \circ \sigma]). \end{aligned}$$

3 Simultaneous Substitution

The presheaf category $[\mathbb{P}, \text{Set}]$ has a non-symmetric monoidal structure, with tensor we denote by $\bullet$. Our aim in this section is to obtain a monoid for $\bullet$ which models simultaneous substitution of linear terms.

Notation 3.1. For $F, G \in [\mathbb{P}, \text{Set}]$ and $n \in \mathbb{N}$, $F \bullet G(n)$ is given as

$$F \bullet G(n) = (\coprod_{k \in \mathbb{N}} Fk \times (\hat{\otimes} G)^k n) / \sim,$$

where $(\hat{\otimes} G)^k$ denotes $G \hat{\otimes} \dots \hat{\otimes} G$, the k -fold tensor product of G , and the relation $\sim$ is defined as the equivalence relation generated by

$$((F\sigma)t; u) \sim (t; R_{\sigma^{-1}}(u)),$$

where $t \in Fk$, $u \in (\hat{\otimes} G)^k n$, and $\sigma \in S_n$. For the definition of $R_{\sigma^{-1}}$, see (2).

Proposition 3.1. *The category $[\mathbb{P}, \text{Set}]$ together with $\bullet$ and evident structural isomorphisms forms a monoidal category with unit given by $\mathcal{Y}_1 : \mathbb{P} \rightarrow \text{Set}$. Moreover, $-\bullet F$ has a right adjoint for all F , exhibiting the category as closed.*

Clearly, the construction of $\bullet$ corresponds to a construction in [6], but our construction uses the linear term-pairing operation $\hat{\otimes}$ in place of finite products as used in that paper. The definition of $\bullet$ also shows that it is equivalent to the substitution operation of Joyal's species [12,13,2].

Definition 3.1. A monoid $X = (X, \mu, \iota)$ in an arbitrary monoidal category $\mathcal{C} = (\mathcal{C}, \cdot, I)$ consists of an object X of $\mathcal{C}$, and two morphisms $\mu : X \cdot X \rightarrow X$, $\iota : I \rightarrow X$ of $\mathcal{C}$ such that the evident diagrams commute. A morphism $f : (X, \mu, \iota) \rightarrow (X', \mu', \iota')$ of monoids is a morphism $f : X \rightarrow X'$ such that $f \circ \mu = \mu' \circ (f \cdot f) : X \cdot X \rightarrow X'$ and $f \circ \iota = \iota' : I \rightarrow X'$. Monoids form a category with initial object $I = (I, I \cdot I \xrightarrow{\sim} I, id_I)$.

Lemma 3.1. 1. There is a natural coherent isomorphism with components $(F \widehat{\otimes} G) \bullet H \rightarrow (F \bullet H) \widehat{\otimes} (G \bullet H)$.
 2. Every element of $G(1)$, equivalently every natural transformation $\mathcal{Y}_1 \rightarrow G$, induces a natural transformation $st_{F,G} : \partial F \bullet G \rightarrow \partial(F \bullet G)$.

In general, if $T : \mathcal{C} \rightarrow \mathcal{C}$ is a strong monad on a monoidal closed category $\mathcal{C} = (\mathcal{C}, \cdot, I)$, the object TI has a monoid structure. Moreover, we can also show that if T is the free monad on a strong endofunctor F on $\mathcal{C}$, the strength of F extends to a strength of T as a monad. Applying this to our Σ , we conclude:

Proposition 3.2. Let Σ be a linear binding signature and let $T\mathcal{Y}_1$ be its free algebra over $\mathcal{Y}_1$. Then $(T\mathcal{Y}_1, \sigma, \eta_{\mathcal{Y}_1})$ is a monoid in $[\mathbb{P}, \text{Set}]$, where $\eta_{\mathcal{Y}_1}$ is the universal arrow, and σ is the unique extension of the unit isomorphism.

Example 3.1. For the case of the linear lambda calculus, $\sigma : \Lambda \bullet \Lambda \rightarrow \Lambda$ is defined as follows: let $(t; u) \in \Lambda \bullet \Lambda(n)$, with $u = [u_1, \dots, u_k, \pi]$. The definition is given by case analysis on t .

$$\begin{aligned} \sigma_n(t; u) &= \text{case } t \text{ of} \\ \text{var}(1) &\quad \Rightarrow u \\ \text{lam}(t') &\quad \Rightarrow \text{lam}(\sigma_{n+1}(t'; [u_1, \dots, u_k, \eta_{\mathcal{Y}_1}(*), \pi \otimes 1])) \\ \text{app}([t_1, t_2, \rho]) &\Rightarrow \text{app}([\sigma_{n_1}(t_1; [u'_1, \dots, u'_i, \tau_1]), \sigma_{n_2}(t_2; [u'_{i+1}, \dots, u'_k, \tau_2]), \pi']), \end{aligned}$$

where for the case of $t = \text{var}(1)$, $k = 1$ and so $u \in \Lambda(n)$; for the case of $t = \text{lam}(t')$, t' is in $\Lambda(k \otimes 1)$; and for the case of **app**, i is the integer such that $0 \leq i \leq k$, $n_1 + n_2 = n$, and $[[u'_1, \dots, u'_i, \tau_1], [u'_{i+1}, \dots, u'_k, \tau_2], \pi']$ is the element of $(\widehat{\otimes} \Lambda)^i \widehat{\otimes} (\widehat{\otimes} \Lambda)^{k-i}(n)$, which is carried to $[u'_1, \dots, u'_k, \pi']$ by the isomorphism of the associativity of $\widehat{\otimes}$, where $[u'_1, \dots, u'_k, \pi'] = R_{\rho^{-1}}([u_1, \dots, u_k, \pi])$. The reordering map $R_{\rho^{-1}}$ is defined in (2).

4 Initial Algebra Semantics

We are now ready to combine the algebra and substitution constructions to obtain the category of models for a linear binding signature. We show that $T\mathcal{Y}_1$, the free Σ -algebra over $\mathcal{Y}_1$ for the linear binding signature Σ , coupled with suitable structure is initial in the category of models. This result indicates that we have a natural combination of the structures for simultaneous substitution and the algebra of linear binding signatures. The result is also consistent with the relationship between our construction and the theory of species.

Definition 4.1. Let F be a strong endofunctor on a monoidal closed category $\mathcal{C} = (\mathcal{C}, \cdot, I)$. An F -monoid $X = (X, \mu, \iota, h)$ consists of a monoid (X, μ, ι) in $\mathcal{C}$ and an F -algebra (X, h) such that the diagram

$$\begin{array}{ccc} F(X) \cdot X & \xrightarrow{st_{X,X}} & F(X \cdot X) \xrightarrow{F\mu} FX \\ h \cdot id_X \downarrow & & \downarrow h \\ X \cdot X & \xrightarrow{\mu} & X \end{array}$$

commutes. F -monoids form a category, with a morphism of F -monoids defined as a morphism of $\mathcal{C}$ which is both an F -algebra and a monoid homomorphism.

From Proposition 3.2, $T\mathcal{Y}_1 = (T\mathcal{Y}_1, \sigma, \eta_{\mathcal{Y}_1})$ is a monoid in $[\mathbb{P}, Set]$, and if we let $\phi_{\mathcal{Y}_1} : \Sigma T\mathcal{Y}_1 \rightarrow T\mathcal{Y}_1$ be a free algebra over $\mathcal{Y}_1$, we have the following result:

Theorem 4.1. Let $T\mathcal{Y}_1$ be a free algebra over $\mathcal{Y}_1$ for a linear binding signature Σ . Then $T\mathcal{Y}_1 = (T\mathcal{Y}_1, \sigma, \eta_{\mathcal{Y}_1}, \phi_{\mathcal{Y}_1})$ is an initial Σ -monoid.

The initial algebra semantics of a Σ -monoid $\mathcal{M}$ is the unique morphism $T\mathcal{Y}_1 \rightarrow \mathcal{M}$ from the initial Σ -monoid to $\mathcal{M}$.

For an example of the above, consider our leading example of the linear lambda calculus as we have developed it through the course of the paper in Examples 2.1 to 2.5 and Example 3.1. The structure of this section for the linear lambda calculus is given by a routine combination of the structures we have developed in previous sections.

Observe here that, in contrast to Fiore et al.[6], we have only considered simultaneous substitution. The construction for single-variable substitution in [6], which worked for non-linear binders, cannot be applied to linear binders. First of all, the axiomatisation of single-variable substitution introduced in that paper is not valid for linear binding. Moreover, in our construction, context extension ∂ does not distribute over $\widehat{\otimes}$, which means that their construction for substitution algebra cannot be applied to our case. These might indicate that the construction for single-variable substitution in [6] may be less general in some subtle way, or that there may be an intrinsic difference between the two styles of substitution. See [20] for further details on single-variable substitution and related issues.

Acknowledgements. I would like to thank Masahiko Sato and Masahito Hasegawa for introducing me to the topic, and I am also very grateful to John Power for his valuable comments.

References

1. A. Barber, P. Gardner, M. Hasegawa, and G. Plotkin. From action calculi and linear logic. In *Computer Science Logic '97 Selected Papers*, volume 1414 of Lecture Notes in Computer Science, pages 78–97, 1998.
2. F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial species and tree-like structures*, volume 67 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 1998.

3. A. Barber, G. Plotkin. Dual intuitionistic linear logic. Submitted.
4. N. de Bruijn. Lambda calculus notations with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae*, 34:381–391, 1972.
5. B.J. Day. On closed categories of functors. In *Midwest Category Seminar Reports IV*, volume 137 of *Lecture Notes in Mathematics*, pages 1–38, 1970.
6. M. Fiore, G. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proceedings of 14th Symposium on Logic in Computer Science*, pages 193–202, IEEE Computer Society Press, 1999.
7. M. Fiore, G. Plotkin. An axiomatisation of computationally adequate domain theoretic model of FPC. In *Proceedings of 9th Symposium on Logic in Computer Science*, pages 92–102, IEEE Computer Society Press, 1994.
8. M. Gabbay, A. Pitts. A new approach to abstract syntax involving binders. In *Proceedings of 14th Symposium on Logic in Computer Science*, pages 214–224, IEEE Computer Society Press, 1999.
9. R. Gordon, J.A. Power. Enrichment through variation. *Journal of Pure and Applied Algebra*, 120:167–185, 1997.
10. M. Hasegawa. Logical predicates for intuitionistic linear type theories. In *Typed Lambda Calculi and its Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 198–212, 1999.
11. G.B. Im, G.M. Kelly. A universal property of the convolution monoidal structure. *J. of Pure and Applied Algebra*, 43:75–88, 1986.
12. A. Joyal. Une théorie combinatoire des séries formelles. *Advances in Mathematics*, 42:1–82, 1981.
13. A. Joyal. Foncteurs analytiques et espèces de structures. In *Combinatoire énumérative*, volume 1234 of *Lecture Notes in Mathematics*, pages 126–159, Springer-Verlag, 1987.
14. J. McCarthy. Towards a mathematical science of computation. In *IFIP Congress 1962*, North-Holland, 1963.
15. E. Moggi. Computational lambda-calculus and monads. In *Proceedings of 4th Symposium on Logic in Computer Science*, pages 14–23, IEEE Computer Society Press, 1989.
16. P.W. O’Hearn, J.C. Reynolds. From Algol to polymorphic linear lambda-calculus. *Journal of the ACM*, January 2000, Vol. 47 No. 1.
17. P. W. O’Hearn, R. D. Tennent. ed. *Algol-like languages*, In *Progress in Theoretical Computer Science*, Birkhauser, 1997.
18. P. W. O’Hearn. A model for Syntactic Control of Interference. *Mathematical Structures in Computer Science*, 3:435–465, 1993.
19. C. Strachey. The varieties of programming language. In *Proceedings of the International Computing Symposium*, pages 222–233. Cini Foundation, Venice, 1972.
20. M. Tanaka. Abstract syntax and variable binding for linear binders (extended version). Draft, 2000.
21. D. Turi, G. Plotkin. Towards a mathematical operational semantics. In *Proceedings of 12th Symposium on Logic in Computer Science*, pages 280–291, IEEE Computer Society Press, 1997.

Regularity of Congruential Graphs

Tanguy Urvoy
turvoy@irisa.fr

IRISA, Campus de Beaulieu,
35042 Rennes, France

Abstract. The aim of this article is to make a link between the *congruential systems* investigated by Conway and the infinite graphs theory. We compare the graphs of congruential systems with a well known family of infinite graphs: the *regular graphs of finite degree* considered by Muller and Shupp, and by Courcelle. We first consider *congruential systems* as word rewriting systems to extract some subfamilies of congruential systems, the *q-p-congruential systems*, representing the regular graphs of finite degree. We then prove the non-regularity of the Collatz's graph.

Introduction

The $3x + 1$ problem concerns iteration of the map $f : \mathbb{N} \longrightarrow \mathbb{N}$, where

$$f(n) = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ \frac{3n+1}{2} & \text{if } n \text{ is odd} \end{cases}$$

The exact origin of the conjecture according to which for any $n > 0$, there is a finite number k of iterations such that $f^k(n) = 1$, is obscure. It is traditionally credited to Lothar Collatz in 1932. The proof of this conjecture seems to be a really intractable open problem (see [7]). We do not have any pretense to give a solution to this problem, we just want to see it as a pathologic case in computer program verification: a simple program for a complicated problem.

In terms of graphs, the Collatz problem can be seen as a reachability problem on the graph $G = \{ n \longrightarrow f(n) \mid n \in \mathbb{N} \}$ of the function f . It can be reformulated by a closed formula of *monadic second order*.

We will consider here the transition graphs of labelled congruential systems, called *the congruential graphs*. It has been proved by Conway [5] that the termination problem of the congruential functions is undecidable. On the other hand, Courcelle [6], has shown that the *regular graphs* have a decidable *monadic second order theory*: any closed monadic second order sentence can be automatically proved for those graphs. Originally, it has been proved by Muller and Shupp [9] that the transition graphs of *pushdown automata* have a decidable second order monadic theory. Then it has been shown by Caucal [3] that *pushdown automata* and *prefix rewriting* are both *internal representations* of regular graphs of finite degree.

In this paper, our aim in studying congruential systems is to get some decidability results: we extract several subfamilies whose transition graphs are exactly the regular graphs of finite degree (c.f. Theorem 3)

We present a restriction according to the degrees of a graph that preserve its regularity. We use this property to prove the non-regularity of the Collatz graph (c.f. Theorem 4)

Preliminaries

Henceforth A will design an alphabet (*i.e.* a finite set of symbols) to label graphs. A *graph* G is a subset of $V \times A \times V$ where V is an arbitrary set. Any $(s, a, t) \in G$ is a *labelled arc* of *source* s , of *target* t , with *label* a and is identified by a labelled transition $s \xrightarrow[G]{a} t$ or directly $s \xrightarrow{a} t$ when G is understood. The free monoid A^*

generated by A is the set of words over A . We write $s \xrightarrow{u} t$ the existence of a path from s to t labelled by word $u \in A^*$; we simply write $s \xrightarrow{*} t$ the existence of a path from s to t . We denote by $V_G = \{s \mid \exists t, a \ s \xrightarrow{a} t \vee t \xrightarrow{a} s\}$ the set of *vertices* of G . The *image* of G by an application h of domain V_G is the graph $h(G) = \{h(s) \xrightarrow{a} h(t) \mid s \xrightarrow[G]{a} t\}$. Recall that a *graph morphism* from a graph G into a graph H is an application $h : V_G \rightarrow V_H$ such that $h(G) \subseteq H$. A *graph isomorphism* h from G to H is a bijective application from V_G to V_H such that $h(G) = H$.

For any vertex s of a graph G , its *out-degree* $d_G^+(s) = |\{t \mid s \rightarrow t\}|$ is the number of its successors, its *in-degree* $d_G^-(s) = |\{t \mid t \rightarrow s\}|$ is its number of predecessors, and its *degree* is $d_G(s) = d_G^+(s) + d_G^-(s)$. The *degree of a graph* is the maximum degree of its vertices.

1 Congruential Systems

1.1 Definition

The *congruential systems* are a non-deterministic generalization of the congruential (or partially linear) functions presented by [5] and [2].

Definition 1. A *congruential system* C is a finite set of rules of the form

$$(p, r) \xrightarrow{a} (q, s)$$

where $a \in A$ and $p, q, r, s \in \mathbb{N}$ with $r < p$ and $s < q$.

The graph $G(C)$ of any congruential system C is defined by

$$G(C) = \left\{ pn + r \xrightarrow{a} qn + s \mid (p, r) \xrightarrow{a} (q, s) \wedge n \in \mathbb{N} \right\}$$

We will call such a graph a *congruential graph*. Note that any congruential graph is of finite degree. Each rule $(p, r) \xrightarrow{a} (q, s)$ will be represented as follows:

$$pn + r \xrightarrow{a} qn + s$$

where p, q are called *periods* and r, s are called the *remainders*.

1.2 Examples

In this section, we discuss some examples of congruential systems and their associated graphs.

Example 1. The rules $\begin{cases} 2n \xrightarrow{a} n \\ 2n + 1 \xrightarrow{b} 3n + 2 \end{cases}$ define the Collatz's graph. The conjecture claims that for any vertex $n > 0$ of this graph, there is a path from n to 1 in the graph. In Example 5, we give a monadic second order sentence to express this conjecture.

Example 2. The rules $\begin{cases} n \xrightarrow{a} 2n \\ n \xrightarrow{b} 2n + 1 \end{cases}$ for $n > 0$, define the complete binary tree. We will see later that this tree is regular.

Example 3. The graph generated by the rules $\begin{cases} n \xrightarrow{a} 2n \\ n \xrightarrow{b} 3n \end{cases}$ restricted to the integers of the form $2^p 3^q$ with $p, q \in \mathbb{N}$, is a grid. The grid has an undecidable monadic second order logic (such a graph is not regular).

1.3 Computing Power of Congruential Systems

Conway has shown that congruential systems have the same computational power as Turing machines, by simulation of any Minsky machine by a congruential function (see [5] or [1] for more details).

One consequence of this result is the undecidability for congruential systems, of any non trivial problem like the termination.

2 Regular Graphs

In this section, we recall the notion of transition systems, regular graphs and monadic second order logic. We also recall the internal representation of regular graphs with rewriting systems needed in section 3.

2.1 Transition Systems and Representation of Infinite Graphs

We will focus on *rewriting systems* because they are closed to congruential systems.

Let us first recall the notion of a *rewriting system*. Let N be an alphabet and N^* be the free monoid generated by N : $N^* = \bigcup_{k \geq 0} N^k$. A labelled rewriting system R on N^* is a finite set of rules of the form $u \xrightarrow{a} v$ with $u, v \in N^*$ and $a \in A$. The *prefix rewriting* $R \cdot N^*$ of such a system is the graph of its prefix transitions *i.e.*

$$R \cdot N^* = \left\{ uw \xrightarrow{a} vw \mid u \xrightarrow{a} v \in R \wedge w \in N^* \right\}$$

Example 4. The following rewriting system:

$$\begin{cases} \epsilon \xrightarrow{a} c \\ \epsilon \xrightarrow{b} d \end{cases}$$

defines by prefix rewriting the complete binary tree:

$$\{u \xrightarrow{a} cu \mid u \in \{c, d\}^*\} \cup \{u \xrightarrow{b} du \mid u \in \{c, d\}^*\}$$

We study properties of a graph up to isomorphism. Note that we can both describe the complete binary tree on the integers ($B \subseteq \mathbb{N} \times \{a, b\} \times \mathbb{N}$) with a congruential system (see Example 2), or on a set of words with a prefix *rewriting system* (see Example 4). The representations of a graph with named vertices is called an *internal* representations. The property of regularity is an *external* property of a graph: it only depends on the structure of the graph up to isomorphism.

Definition 2. [6] *A regular graph is a graph generated by a deterministic graph grammar from a finite hypergraph.*

2.2 Second Order Logic and Regularity

To build the monadic second order formulas, we take two disjoint sets: a set of variables of vertices and a set of variables of vertex sets. An atomic formula is one of the two following forms:

$$x \in X \quad \text{or} \quad x \xrightarrow{a} y$$

where X is a vertex set variable, x and y are vertex variables, and $a \in A$. From those atomic formulas, we construct the monadic second order formulas with the propositional connectors $\neg$, $\wedge$ and the existential quantifier $\exists$ ranging on the two types of variables. A closed formula is a formula without free variable. The set of closed formulas that are true for a given graph is called its *monadic theory*. As usual, it is possible to express the propositional connectors like $\vee, \Rightarrow$, and the universal quantifier $\forall$ ranging on the two types of variables:

$$\varphi \vee \psi : \neg(\neg\varphi \wedge \neg\psi); \varphi \Rightarrow \psi : \neg(\varphi \wedge \neg\psi); \forall X \varphi : \neg(\exists X \neg\varphi)$$

Example 5. We define the equality and the transitive closure:

$$\begin{array}{ll} x = y & : \forall X (x \in X \Leftrightarrow y \in X) \\ x \xrightarrow{*} y & : \forall X ((x \in X \wedge \text{Closed}(X)) \Rightarrow y \in X) \\ \text{with } \text{Closed}(X) & : \forall x \forall y ((x \in X \wedge x \rightarrow y) \Rightarrow y \in X) \end{array}$$

This permit to express the Collatz problem:

$$\exists x \exists y (x \xrightarrow{b} y \wedge y \xrightarrow{a} x \wedge \neg(x = y) \wedge \forall z (\neg(z \xrightarrow{a} z) \Longrightarrow z \xrightarrow{*} x))$$

We can decide any such monadic sentence on any regular graph.

Theorem 1. [6] *Any regular graph has a decidable monadic second order theory.*

2.3 Internal Representation with Rewriting Systems

We will see that there is a close relationship between congruential systems and rewriting systems. To study the regularity of congruential graphs, we will use an internal characterization of finite degree regular graphs with rewriting systems.

We first recall some definitions. The *restriction* $G|_L$ of a graph $G \subseteq V \times A \times V$ by a set $L \subseteq V$, is the graph

$$G|_L = \left\{ u \xrightarrow{a} v \mid u \xrightarrow[G]{a} v \wedge u, v \in L \right\}$$

If V is a free monoid and if L is a rational language (*i.e.* a subset of V recognized by a finite automaton) $G|_L$ is a *rational restriction* of G .

Theorem 2. [4] *The regular graphs of finite degree are (up to isomorphism) the rational restrictions of prefix transition graphs.*

3 Regular Congruential Graphs

In this section we extract families of congruential systems representing effectively the regular graphs of finite degree. There is a close relationship between congruential systems and rewriting systems. To study congruential graphs, we will code congruential systems with rewriting systems. For this purpose we use the p -ary representation of positive integers. Let us recall shortly this representation of integers. Let $p > 1$ be an integer called the *basis* and let $\llbracket 0, p \rrbracket = \{0, \dots, p-1\}$. We associate to each word $w = w_1 \dots w_{|w|}$ of $\llbracket 0, p \rrbracket^*$ its value $[w]_p = \sum_{i=1}^{|w|} w_i p^{i-1}$. The mapping $w \mapsto [w]_p$ cannot be inverted due to the ambiguity introduced by the leadings zeros. However its restriction to the set $(\mathcal{N})_p = \llbracket 0, p \rrbracket^* - \llbracket 0, p \rrbracket^* 0$ of *normalized* words of $\llbracket 0, p \rrbracket^*$, is one-to-one. Given an integer n , the unique w in $(\mathcal{N})_p$ such that $[w]_p = n$ is denoted by $(n)_p$ and is called the *normal p -ary representation* of n . In particular $(0)_p = \epsilon$ and we have $([u]_p + p^{|u|}n)_p = u(n)_p$ for any $u \in \llbracket 0, p \rrbracket^*$ and $n > 0$.

Definition 3. (See [10]) A set $B \subseteq \mathbb{N}$ is called p -recognizable for $p > 1$ if the set $(B)_p = \{(n)_p \mid n \in B\}$ is a rational subset of $\llbracket 0, p \rrbracket^*$. It is called 1-recognizable or simply recognizable if $\{0^n \mid n \in B\}$ is a recognizable subset of $\{0\}^*$: B is a finite union of sets of the form $\{an + b \mid n \geq 0\}$.

For any $L \subseteq \llbracket 0, p \rrbracket^*$, we denote by $[L]_p = \{[w]_p \mid w \in L\}$ the set of integers coded by L in base p .

Definition 4. Let $p, q > 0$. A q - p -congruential system is a congruential system with rules of the form

$$(qp^k, r) \xrightarrow{a} (qp^l, s)$$

A q - p -congruential graph is the graph of a q - p -congruential system.

Proposition 1. Any p -recognizable restriction of a q - p -congruential graph is regular.

Proof. To each integer $n = qm + r$ with $0 \leq r < q$ we associate the word $(n)_{q,p} = r \cdot (m)_p$. Let B be a p -recognizable set and C be a q - p -congruential system.

First note that $(B)_{q,p}$ is rational because $(B)_{q,p} = \bigcup_{0 \leq r < q} r \cdot (B_{q,r})_p$ where $B_{q,r} = \{n \div q \mid n \in B \wedge n \bmod q \equiv r\}$ is p -recognizable (the Euclidean division by a constant preserves the p -recognizability of sets [10]).

Consider the rewriting system

$$(C)_{q,p} = \left\{ (r)_{q,p} 0^{1+k-|(r)_{q,p}|} \xrightarrow{a} (s)_{q,p} 0^{1+l-|(s)_{q,p}|} \mid (qp^k, r) \xrightarrow{a}_C (qp^l, s) \right\}$$

Let μ be the greatest period of C then $G(C)_{|B-\llbracket 0, \mu \rrbracket}$ is isomorphic by $(\)_{q,p}$ to the prefix rewriting graph $(C)_{q,p} \cdot \llbracket 0, p \rrbracket^*$ restricted to $(B)_{q,p}$. This proves the regularity of $G(C)_{|B}$ by Theorem 2. $\square$

Example 6. Construction of a prefix rewriting graph from a 1-2-congruential system. Consider the 1-2-congruential system C of Example 2. Its maximum period μ is 2. Let $L = (\mathbb{N} - \llbracket 0, 2 \rrbracket)_{1,2} = (\mathbb{N})_{1,2} - \{0, 01\}$ and $(C)_{1,2}$ be the rewriting system image of C by $(\)_{1,2}$.

$$(C)_{1,2} : \begin{cases} (0)_{1,2} 0^{1+0-|(0)_{1,2}|} \xrightarrow{a} (0)_{1,2} 0^{1+1-|(0)_{1,2}|} \\ (0)_{1,2} 0^{1+0-|(0)_{1,2}|} \xrightarrow{b} (1)_{1,2} 0^{1+1-|(1)_{1,2}|} \end{cases} = \begin{cases} 0 \xrightarrow{a} 00 \\ 0 \xrightarrow{b} 01 \end{cases}$$

$((C)_{1,2} \cdot \llbracket 0, 2 \rrbracket^*)_{|L}$ is isomorphic to $G(C)_{|\mathbb{N}-\llbracket 0, 2 \rrbracket}$ and $G(C)_{|\llbracket 0, 2 \rrbracket}$ is finite (See Fig. 1).

Theorem 3. Let $p \geq 2$ and $q \geq 1$. The p -recognizable restrictions of the q - p -congruential graphs are exactly and effectively the regular graphs of finite degree.

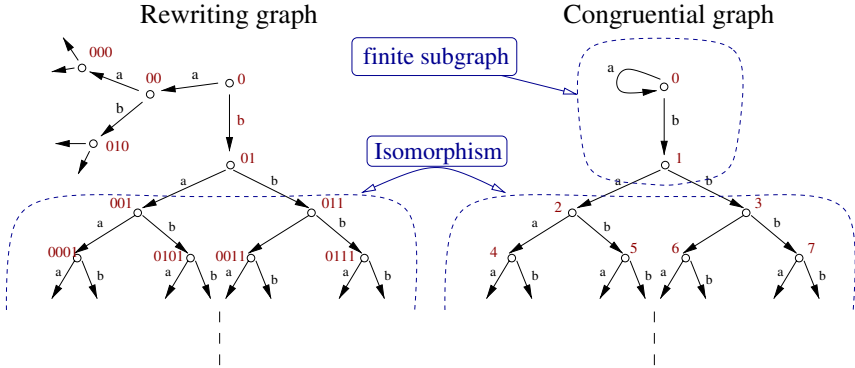


Fig. 1. Construction of a prefix rewriting graph from a congruential one.

Proof. By Theorem 2, any regular graph of finite degree G is isomorphic to a rational restriction of a prefix rewriting graph. Let L be a rational language of A^* and R be a rewriting system on A such that $(R \cdot A^*)|_L$ is isomorphic to G . Let c be an injective function from A into $\{0, 1\}^{|A|-1} \cdot 1$. We extend c by morphism from A^* into $(\{0, 1\}^{|A|-1} \cdot 1)^*$. Let G' be the image of $(R \cdot A^*)|_L$ by c : $G' := c((R \cdot A^*)|_L) = (c(R) \cdot \{0, 1\}^*)|_{c(L)} = (c(R) \cdot \llbracket 0, p \rrbracket^*)|_{c(L)}$. Then G' is isomorphic to G .

Consider the q - p -congruential system

$$C := q[c(R)]_p = \left\{ (qp^{|u|}, q[u]_p) \xrightarrow{a} (qp^{|v|}, q[v]_p) \mid u \xrightarrow{a} v \in c(R) \right\}$$

and the p -recognizable set $q[c(L)]_p = \{q[w]_p \mid w \in c(L)\}$. The graph $G(C)|_{q[c(L)]_p}$ is the same as $q[G']_p$ hence is isomorphic to $[G']_p$. $\square$

Example 7. Construction of a 2-3-congruential system from a regular graph of finite degree. The rewriting system R generates a regular graph G by prefix rewriting and restriction to $L = a^* + ba^* + da^*$.

$$R : \begin{cases} \epsilon \xrightarrow{a} a \\ \epsilon \xrightarrow{b} b \\ ba \xrightarrow{d} d \\ d \xrightarrow{d} b \end{cases} \longrightarrow c(R) : \begin{cases} \epsilon \xrightarrow{a} 001 \\ \epsilon \xrightarrow{b} 011 \\ 011001 \xrightarrow{d} 111 \\ 111 \xrightarrow{d} 011 \end{cases}$$

The alphabet used to code the vertices is $A = \{a, b, d\}$. We use the coding function c defined by $c(a) = 001$, $c(b) = 011$ and $c(d) = 111$ to avoid the side effect of leading zeros. We have $c(L) = (001)^* + 011(001)^* + 111(001)^*$. The image of R by c gives by prefix rewriting and restriction to $c(L)$, the graph $G' = (c(R) \cdot \llbracket 0, 3 \rrbracket^*)|_{c(L)}$ which is isomorphic to G (See Fig. 2).

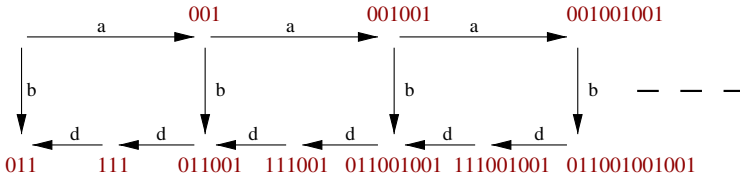


Fig. 2. The graph G' .

We can construct the 3-recognizable set $B = 2 \cdot [c(L)]_3$ and the 2-3-congruential system C to obtain the graph $G(C)|_B = [G']_p$, which is isomorphic to G :

$$C : \begin{cases} 2n \xrightarrow{a} 54n + 18 \\ 2n \xrightarrow{b} 54n + 24 \\ 1458n + 510 \xrightarrow{d} 54n + 26 \\ 54n + 26 \xrightarrow{d} 54n + 24 \end{cases}$$

Theorem 3 does not give a necessary condition for the regularity of a congruential graph.

4 Non Regularity of the Collatz Graph

After having studied families of congruential graphs representing the regular graphs of finite degree, it is natural to know whether the Collatz graph is regular. We show here that the Collatz graph is not regular in the following general sense: whatever the labelling and the orientation of the arcs you choose for this graph, it will remain non regular.

4.1 Necessary Conditions for Regularity

There is no criterion to decide in general whether a graph is regular or not, but there are some properties that gives necessary conditions:

Proposition 2. [4] *Every regular graph has a finite number of nonisomorphic connected components.*

We give here another necessary condition:

Proposition 3. *If G is a regular graph and D a finite set of integers, then $G|_{\{s \in V_G \mid d_G(s) \in D\}}$ is a regular graph.*

By building a grammar generating $G|_{\{s \in V_G \mid d_G(s) \in D\}}$ from the grammar of G by computing the degrees (finite or infinite) of vertices in the graph grammar with a fixed point algorithm. The convergence is ensured by the fact that a regular graph only admits a finite number of degrees.

4.2 Non Regularity of the Collatz's Graph

Let us study a particular family of vertices of the Collatz graph: let $K_n := K_n^1 \cup K_n^2$ where $K_n^1 := \left\{ \frac{3^{2n}-1}{2}, 2^{2n}-1 \right\} \cup \{2k \mid k \in K_n^2\}$ and $K_n^2 := \{2^k 3^{2n-k} - 1 \mid k \in \llbracket 0, 2n \rrbracket\}$

Lemma 1. *For any $q, p \geq 0$, we have $2^{p+1}3^q - 1 \xrightarrow[G(C)]{b} 2^p3^{q+1} - 1$*

Proof. We have $2^{p+1}3^q - 1 = 2(2^p3^q - 1) + 1$ and we apply the rule $2n+1 \xrightarrow{b} 3n+2$. $\square$

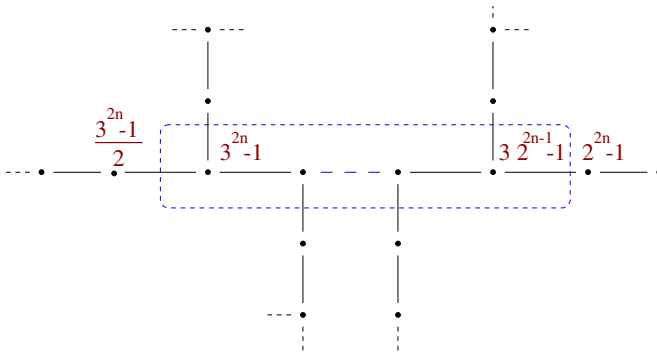


Fig. 3. A Subgraph of the Collatz graph.

Let $G(C)$ be the collatz graph. By Lemma 1, it follows: $2^n - 1 \xrightarrow[G(C)]{b^n} 3^n - 1$ for every $n \geq 1$.

Lemma 2. *For any integer $n > 0$,*

- (i) $G(C)|_{K_n^2}$ is connected;
- (ii) $\forall s \in K_n^1, d_{G(C)}(s) = 2$;
- (iii) $\forall s \in K_n^2, d_{G(C)}(s) = 3$.

Proof. (i) is a consequence of Lemma 1.

First remark that $d_{G(C)}^-(n) = 2 \Leftrightarrow n \equiv 2 \pmod 3$ and $d_{G(C)}^-(n) = 1 \Leftrightarrow n \not\equiv 2 \pmod 3$.

For (ii) we have $\frac{3^{2n}-1}{2} = 1 + 3 \frac{3^{2n-1}-1}{2} \equiv 1 \pmod 3$ As $\forall n \in \mathbb{N}, \frac{3^{2n}-1}{2} \in \mathbb{N}$, we have $2^{2N} - 1 \equiv 0 \pmod 3$. For any $s \in K_n^2$, we have $2s = 3(2^{k+1}3^{2n-k-1}) - 2 \equiv 1 \pmod 3$.

For (iii), we remark that any element s of K_n^2 verifies $s \equiv 2 \pmod 3$. $\square$

Note that K_n^2 is “surrounded” by K_n^1 in the Collatz graph (See Fig. 3).

Theorem 4. *The unlabelled Collatz graph is not regular.*

Proof. Consider the graph $G' := G(C)|_{\{s \in V_{G(C)} \mid d(s)=3\}}$. By Lemma 2, for each value $n > 0$, $G'_{|K_n^2}$ is a connected component of G' of size $2n$: G' has an infinity of non isomorphic connected components, hence is not regular. By Proposition 3, the Collatz graph is not regular. $\square$

5 Conclusion

Classification of infinite graphs allows us to study a lot of different discrete systems in a unifying way. We have extracted here families of congruential graphs, the q - p -congruential graphs, that are regular. We know for those graphs that any problem similar to the Collatz conjecture is automatically provable. However, the non regularity of the Collatz graph confirms that the structure of this graph is complex but it does not prove that its monadic second order theory is undecidable. We can prove that the Collatz graph is a *rational graph* [8], but there is no decision result in general for this family of graphs.

Acknowledgement. Let me thank firstly D. Caucal without whom this article would never have existed. I also thank S. Burckel, T. Cachat, T. Knapik, C. Morvan, V. Schmitt and some anonymous referees for their numerous remarks.

References

1. S. Burckel. Systèmes congruentiels. Technical report, *Séminaire de logique et algorithmique*. Université de Rouen, 1992.
2. S. Burckel. Functional equations associated with congruential functions. *Theoretical Computer Science*, 123(2):397–406, 1994.
3. D. Caucal. On the regular structure of prefix rewriting. *Theoretical Computer Science*, 106:61–86, 1992.
4. D. Caucal. Bisimulation of context-free grammars and of pushdown automata. *CSLI Modal logic and process algebra*, 53:85–106, 1995.
5. J.H. Conway. Unpredictable iterations. In *Number Theory*, pages 49–52, 1972.
6. B. Courcelle. Graph rewriting, an algebraic and logic approach. In J. Van Leuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 193–242, 1990. Elsevier.
7. J. C. Lagarias. The $3x+1$ problem and its generalizations. *The American Mathematical Monthly*, 92(1):3–23, 1985.
8. C. Morvan. On rational graphs. *Fossacs 2000*, 2000.
9. D. E. Muller and P. E. Schupp. The theory of ends, pushdown automata, and second-order logic. *Theoretical Computer Science*, 1985.
10. D. Perrin. Finite automata. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 1–53. North Holland, 1990.
11. Shallit and Wilson. The “ $3x+1$ ” problem and finite automata. *BEATCS: Bulletin of the European Association for Theoretical Computer Science*, 46, 1992.

Sublinear Ambiguity

Klaus Wich

Institut für Informatik, Universität Stuttgart,
Breitwiesenstr. 20-22, 70565 Stuttgart.
E-mail: wich@informatik.uni-stuttgart.de

Abstract. A context-free grammar G is ambiguous if there is a word that can be generated by G with at least two different derivation trees. Ambiguous grammars are often distinguished by their degree of ambiguity, which is the maximal number of derivation trees for the words generated by them. If there is no such upper bound G is said to be ambiguous of infinite degree. By considering how many derivation trees a word of at most length n may have, we can distinguish context-free grammars with infinite degree of ambiguity by the growth-rate of their ambiguity with respect to the length of the words. It is known that each cycle-free context-free grammar G is either exponentially ambiguous or its ambiguity is bounded by a polynomial. Until now there have only been examples of context-free languages with inherent ambiguity $2^{\Theta(n)}$ and $\Theta(n^d)$ for each $d \in \mathbb{N}_0$. In this paper first examples of (linear) context-free languages with nonconstant sublinear ambiguity are presented.

1 Introduction

A context-free grammar G is ambiguous if there is a word that can be generated by G with at least two different derivation trees. Ambiguous grammars are often distinguished by their degree of ambiguity, which is the maximal number of derivation trees for the words generated by them. If there is no such upper bound G is said to be ambiguous of infinite degree.

In [5] and [6] the ambiguity function has been introduced as a new tool for examining the ambiguity of cycle-free context-free grammars. The ambiguity function maps the natural number n to the maximal number of derivation trees which a word of at most length n may have. It has been shown there that for cycle-free context-free grammars the ambiguity function is either an element of $2^{\Theta(n)}$ or of $\mathcal{O}(n^d)$ for a $d \in \mathbb{N}_0$ which can be effectively constructed from G .

L has inherent ambiguity $\Theta(f)$ if there is a grammar with an ambiguity function in $\mathcal{O}(f)$ and each grammar that generates L has an ambiguity function in $\Omega(f)$. Languages with inherent ambiguity $2^{\Theta(n)}$ and with inherent ambiguity $\Theta(n^d)$ for each $d \in \mathbb{N}_0$ have been presented in [4].

It is easy to prove that the above mentioned infinite ambiguities are exactly the ones that can occur, of course not inherently, in *right linear* grammars over a *single letter alphabet*. In that sense sublinear ambiguity requires a more complicated structure.

In this article first examples of context-free grammars having sublinear ambiguity are presented (they are even linear). These grammars have logarithmic and square-root ambiguity, respectively. Moreover it is shown that these ambiguities are inherent for the corresponding languages.

2 Preliminaries

Let Σ be a finite alphabet. For a word $w \in \Sigma^*$, a symbol $a \in \Sigma$, and $n \in \mathbb{N}$ the length of w is denoted by $|w|$, the number of a 's in w is denoted by $|w|_a$. The empty word is denoted by ε . The set $\Sigma^{\leq n}$ denotes all words over Σ with length up to n . The cardinality of a set S is denoted by $|S|$.

A *context-free grammar* is a quadruple $G = (N, \Sigma, P, S)$, where N and Σ are finite disjoint alphabets of nonterminals and terminals, respectively, $S \in N$ is the start symbol, and $P \subseteq N \times (N \cup \Sigma)^*$ is a finite set of productions. We usually write $A \rightarrow \alpha$ or $(A \rightarrow \alpha)$ for the pair (A, α) . We write $A \rightarrow \alpha \mid \beta$ as an abbreviation for the two productions $A \rightarrow \alpha$, $A \rightarrow \beta$.

For a context-free grammar $G = (N, \Sigma, P, S)$ and $\alpha, \beta \in (N \cup \Sigma)^*$, we say that α derives β in one step, denoted by $\alpha \Rightarrow_G \beta$, if there are $\alpha_1, \alpha_2, \gamma \in (N \cup \Sigma)^*$ and $A \in N$ such that $\alpha = \alpha_1 A \alpha_2$, $\beta = \alpha_1 \gamma \alpha_2$ and $(A \rightarrow \gamma) \in P$. We say that α derives β leftmost in one step if in the definition above $\alpha_1 \in \Sigma^*$.

Let $\Rightarrow_G^+$ denote the transitive closure of $\Rightarrow_G$, and $\Rightarrow_G^*$ denote the reflexive closure of $\Rightarrow_G^+$. For $\alpha, \beta \in (N \cup \Sigma)^*$ and $\pi \in P^*$ we write $\alpha \Rightarrow_G^\pi \beta$ if α derives β by the sequence of leftmost steps indicated by π . We call π a *parse* from α to β in this case. The language generated by G is defined by $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$. If the grammar is clear from the context, the subscript G is omitted. A language L is said to be *context-free* if there is a context-free grammar G with $L = L(G)$. Let $G = (N, \Sigma, P, S)$ be a context-free grammar, and $\alpha \in (N \cup \Sigma)^*$. We say that α is a *sentential form* if $S \Rightarrow^* \alpha$. The grammar G is said to be *cycle-free* if there is no $A \in N$ such that $A \Rightarrow^+ A$.

Definition 1. Let $G = (N, \Sigma, P, S)$ be a context-free grammar, $w \in \Sigma^*$ and $n \in \mathbb{N}$. We define the *ambiguity* of w , and the *ambiguity function* $am_G : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ as follows:

$$am_G(w) := |\{\pi \in P^* \mid S \Rightarrow_G^\pi w\}|$$

$$am_G(n) := \max\{am_G(w) \mid w \in \Sigma^{\leq n}\}$$

Note that for a grammar G which contains cycles the set $parse_G(A, \beta)$ may be infinite. But for all cycle-free grammars G the ambiguity function am_G is a total mapping $am_G : \mathbb{N}_0 \rightarrow \mathbb{N}_0$. Note that $am_G(w) = 0$ for all $w \notin L(G)$.

Definition 2. Let $f : \mathbb{N}_0 \rightarrow \{\mathbb{R} \mid r > 0\}$ be a total function and L a context-free language. We call L *inherently f -ambiguous* if

- (i) for all context-free grammars G such that $L = L(G)$ we have $am_G = \Omega(f)$,
and
- (ii) there is a grammar G_0 such that $L = L(G_0)$ and $am_{G_0} = \mathcal{O}(f)$.

Note that we have defined inherent complexity *classes* for languages here. Let L be an f -ambiguous context-free language such that $f(n) > 1$ for some $n \in \mathbb{N}$. This does not imply that each grammar G with $L(G) = L$ has a word of at most length n with at least $f(n)$ derivation trees. In fact there are grammars for L which generate all words up to length n unambiguously.

3 Sublinear Languages

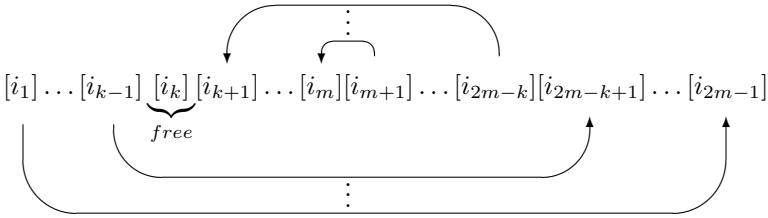
Let $\Sigma = \{0, 1\}$ be an alphabet. We denote $0^{i-1}1$ by $[i]$ for all $i \in \mathbb{N}$. We define the following two languages:

$$L_{\log} := \{[i_1] \dots [i_{2m-1}] \mid m \in \mathbb{N}; i_1, \dots, i_{2m-1} \in \mathbb{N}; \exists 1 \leq k \leq m : \\ ((\forall \ell < k : 2i_\ell = i_{2m-\ell}) \wedge (\forall m \geq \ell > k : i_\ell = 2i_{2m+1-\ell}))\}$$

and analogously

$$L_{\sqrt{}} := \{[i_1] \dots [i_{2m-1}] \mid m \in \mathbb{N}; i_1, \dots, i_{2m-1} \in \mathbb{N}; \exists 1 \leq k \leq m : \\ ((\forall \ell < k : i_\ell + 1 = i_{2m-\ell}) \wedge (\forall m \geq \ell > k : i_\ell = i_{2m+1-\ell} + 1))\}$$

Let $w = [i_1] \dots [i_{2m-1}] \in L_{\log}$ for some $i_1, \dots, i_{2m-1} \in \mathbb{N}$ and some $m \in \mathbb{N}$. For $1 \leq k \leq 2m-1$ we call $[i_k]$ the k -th block of w . The blocks are pairwise correlated from the borders to the middle. One block k ($1 \leq k \leq m$) is not forced to have a correlation. When passing this free block, the direction of the correlation is reversed. For $L_{\log}$ the quotient of the correlated numbers is 2, for $L_{\sqrt{}}$ their difference is 1. The situation is illustrated in the following diagram. Arrows indicate correlations forced by the definition of the language:



The languages $L_{\log}$ and $L_{\sqrt{}}$ are generated by the subsequently defined grammars $G_{\log}$ and $G_{\sqrt{}}$, respectively. For $sub \in \{\log, \sqrt{}\}$ we define

$G_{sub} := (\{A, B, C, D, S\}, \{0, 1\}, P_{sub}, S)$ as follows:

$P_{\log} := \left\{ \begin{array}{l l l} S \rightarrow 1S01 & 0A01 & B \\ A \rightarrow 0A00 & 1S00 & \\ B \rightarrow 0B & 1C & 1 \\ C \rightarrow 01C1 & 00D1 & 011 \\ D \rightarrow 00D0 & 01C0 & 010 \end{array} \right\}$	$P_{\sqrt{}} := \left\{ \begin{array}{l l l} S \rightarrow 1S01 & 0A1 & B \\ A \rightarrow 0A0 & 1S00 & \\ B \rightarrow 0B & 1C & 1 \\ C \rightarrow 01C1 & 0D1 & 011 \\ D \rightarrow 0D0 & 01C0 & 010 \end{array} \right\}$

It is easily verified that these grammars generate the languages defined above. The derivation starts with a (possibly empty) finite number of cycles in the nonterminals S and A which produces the blocks to the left of the free block and the corresponding blocks at the right end of the word. Eventually the production $S \rightarrow B$ is applied. The nonterminal B generates the free block. Finally either the derivation is terminated with the production $B \rightarrow 1$, or with $B \rightarrow 1C$ we begin to produce blocks with the opposite correlation to the right of the free block, by using the nonterminals C and D .

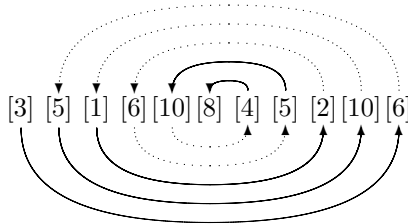
3.1 Sublinear Ambiguity of the Presented Grammars

In this section we prove that $am_{G_{\log}} \in \mathcal{O}(\log n)$. Analogously $am_{G_{\sqrt{\cdot}}} \in \mathcal{O}(\sqrt{n})$ can be shown, which however will not be done here.

Definition 3. Let $i_1, \dots, i_{2m-1} \in \mathbb{N}$ for some $m \in \mathbb{N}$, $w = [i_1] \dots [i_{2m-1}]$, and $1 \leq \ell \leq m$.

- The word w has a forward correlation at block ℓ if and only if $\ell < m$ and $2i_\ell = i_{2m-\ell}$. It has a forward crack if and only if $\ell < m$ and $2i_\ell \neq i_{2m-\ell}$.
- The word w has a backward correlation at block ℓ if and only if $1 < \ell \leq m$ and $i_\ell = 2i_{2m+1-\ell}$. It has a backward crack if and only if $1 < \ell \leq m$ and $i_\ell \neq 2i_{2m+1-\ell}$.
- Block ℓ is isolated in w if and only if block ℓ has neither a forward nor a backward correlation.

Example 1. We illustrate these definitions by the following diagram. The relevant relations between blocks are on a spiral from the leftmost block to the block in the middle, indicated by solid and dotted arrows.



The blocks 1, 2, and 3 have a forward correlation. Block 4 and 5 have a forward crack. Forward correlations and cracks are indicated by solid and dotted arrows from left to right, respectively. Blocks 2, 3, and 4 have a backward crack, blocks 5 and 6 have a backward correlation, again indicated by dotted and solid arrows, this time from right to left. Block 4 is isolated, since it has neither a forward nor a backward correlation.

Definition 4. Let $m, r \in \mathbb{N}$, $i_1, \dots, i_{2m-1} \in \mathbb{N}$, and $w = [i_1] \dots [i_{2m-1}]$.

- $(r * w) := [ri_1] \dots [ri_{2m-1}]$
- $z_m = \begin{cases} [1] & \text{for } m = 1 \\ [1](4 * z_{m-1})[2] & \text{for } m > 1 \end{cases}$
For example $z_4 = [1][4][16][64][32][8][2]$.
- $L_{\text{even},r} := \{(r * z_m) \mid m \in \mathbb{N}\}$
- $L_{\text{even}} := \bigcup_{r \in \mathbb{N}} L_{\text{even},r}$
- $L_{\text{min}} := L_{\text{even},1}$.

Note that $L_{\text{min}} = \{z_m \mid m \in \mathbb{N}\}$, and that a word is in L_{even} if and only if it has no cracks.

For a word with an isolated block we know that this block has to be derived by the nonterminal B and therefore the derivation of the whole word is completely determined. In general cracks provide information about the position of the free block. But the language definition does not require the existence of cracks. Hence $L_{\text{even}} \subseteq L_{\log}$. For a word $w \in L_{\text{even}}$ any block up to the one in the middle can be produced by nonterminal B . For example in the word $z_3 = [1][4][16][8][2]$ either $[1]$, $[4]$, or $[16]$ is the free block. This gives ambiguity 3. Hence, for each $m, r \in \mathbb{N}$, the word $(r * z_m)$ has m derivations. Moreover we will prove that z_m is the shortest word in $L_{\log}$ with m derivations, which inspired the name L_{min} . Due to the free block the forward and backward correlations are interlocked. Therefore in a word without cracks the length of the blocks is strictly increasing along the spiral, while the ambiguity is proportional to the number of blocks. Thus the ambiguity is sublinear.

Lemma 1. *Let $w \in L_{\log}$ and $w \notin L_{\text{min}}$. Then there is a word $w' \in \Sigma^*$ with $|w'| < |w|$ and $\text{am}_{G_{\log}}(w') = \text{am}_{G_{\log}}(w)$.*

Proof. We distinguish three cases.

Case 1: $w \in L_{\text{even}}$.

For some $m, r \in \mathbb{N}$ we have $w = (r * z_m)$. Since $w \notin L_{\text{min}}$ we have $r > 1$. Thus we obtain $|z_m| < r|z_m| = |(r * z_m)| = |w|$. Moreover $\text{am}_{G_{\log}}(z_m) = m = \text{am}_{G_{\log}}((r * z_m)) = \text{am}_{G_{\log}}(w)$.

Case 2: w has a block ℓ with a forward crack.

For some $m \in \mathbb{N}$ we have $|w|_1 = 2m - 1$, which is the number of blocks in w . Since block ℓ has a forward crack, by definition $\ell < m$. Moreover block ℓ cannot be generated by the nonterminals S and A . Therefore block ℓ is either produced by nonterminal B or by the nonterminals C and D . In both cases blocks $\ell + 1$ up to block m are generated by C and D . Since $\ell < m$ there is at least one such block. But then the derivation after generating block ℓ is completely determined by the blocks $\ell + 1$ up to block m . That is, by erasing these and their correlated blocks we obtain a word w' which consists of $2\ell - 1$ blocks from w , and which has the same ambiguity as w . Hence we obtain $|w'| < |w|$ and $\text{am}_{G_{\log}}(w') = \text{am}_{G_{\log}}(w)$.

Case 3: w has a block ℓ with a backward crack.

For some $m \in \mathbb{N}$ we have $|w|_1 = 2m - 1$. Since block ℓ has a backward crack, by definition $\ell > 1$. Moreover block ℓ cannot be generated by the nonterminals C and D . Therefore block ℓ is either produced by nonterminal B or by the

nonterminals S and A . In both cases blocks 1 up to block $\ell - 1$ are generated by S and A . Since $\ell > 1$, there is at least one such block. But then the derivation until generating block ℓ is completely determined by the blocks 1 up to block $\ell - 1$. That is, by erasing these and their correlated blocks we obtain a word w' which consists of $2(m - \ell) + 1$ blocks from $|w|$ and which has the same ambiguity as w . Hence we obtain $|w'| < |w|$ and $am_{G_{\log}}(w') = am_{G_{\log}}(w)$.

Theorem 1. $\forall j \in \mathbb{N} \forall w \in L_{\log} : \quad |w| < |z_j| \text{ implies } am_{G_{\log}}(w) < j$

Proof. Let w be a shortest word such that $am_{G_{\log}}(w) \geq j$. Since $am_{G_{\log}}(z_j) = j$ we observe that $|w| \leq |z_j|$. Moreover Lemma 1 implies that w is in L_{min} and hence $w = z_i$ for some $i \in \mathbb{N}$. Since $am_{G_{\log}}(z_i) = i$ we get $i \geq j$. Now $|z_i| = |w| \leq |z_j|$ implies $i \leq j$. Thus we obtain $i = j$, that is $w = z_j$ which proves Theorem 1.

By Theorem 1 we obtain the following table

ambiguity	shortest word	length
1	$z_1 = [1]$	1
2	$z_2 = [1][4][2]$	7
3	$z_3 = [1][4][16][8][2]$	31
$\vdots$	$\vdots$	$\vdots$
i	$\dots$	$\frac{1}{2}4^i - 1$

If we proceed analogously for $L_{\sqrt{\cdot}}$ we obtain

Theorem 2.

$$am_{G_{\log}}(n) = \lfloor \log_4(2n + 2) \rfloor = \mathcal{O}(\log n)$$

$$am_{G_{\sqrt{\cdot}}}(n) = \left\lfloor \frac{1}{4} + \sqrt{\frac{1}{2}n + \frac{1}{16}} \right\rfloor = \mathcal{O}(\sqrt{n})$$

3.2 Inherence of the Sublinearity

In this section we will prove that the language $L_{\log}$ has inherent logarithmic ambiguity. We already proved that logarithmic ambiguity is sufficient to generate the language. Thus we have to prove that less than logarithmic ambiguity does not suffice. First we prove a technical lemma.

Lemma 2. *Let $w = [i_1] \dots [i_{2m-1}]$ for some $m \in \mathbb{N}$ and $i_1, \dots, i_{2m-1} \in \mathbb{N}$, and let $1 \leq n \leq \frac{1}{3}(m - 1)$. Then*

$$i_{m-3n} = i_{m+3n} \text{ and } i_m = i_{m+2n} \text{ and } i_{m+1} = i_{m+1-2n} \text{ implies } w \notin L_{\log}.$$

Proof. By definition w has a forward crack at block $m - 3n$. Now assume $w \in L_{\log}$. Then all blocks numbered $m - 3n + 1$ up to m must have a backward correlation. In particular $i_{m+1-2n} = 2i_{m+2n}$ and $i_m = 2i_{m+1}$. But then $i_m = 2i_{m+1} = 2i_{m+1-2n} = 4i_{m+2n} = 4i_m$ is a contradiction.

The lemma above is important because it tells us that in a word of $L_{\log}$ a sequence consisting of $2n$ blocks cannot be repeated too often in the vicinity of the middle block.

Theorem 3. $L_{\log}$ has inherent logarithmic ambiguity.

Proof. Let $G = (N, \Sigma, P, S)$ be an arbitrary context-free grammar such that $L(G) = L_{\log}$. We will apply Ogden's iteration lemma for context-free grammars (see [1, Lemma 2.5]). Let p be the constant of Ogden's iteration lemma for G . We define $s := p + 1$ and $r := s! + s$. For each $m \in \mathbb{N}$, and $1 \leq n \leq 2m - 1$, we define $i_{m,n}$ such that $[i_{m,n}]$ is the n -th block of z_m . Let

$$S_m := \{[ri_{m,1}] \dots [ri_{m,\ell-1}][si_{m,\ell}][ri_{m,\ell+1}] \dots [ri_{m,2m-1}] \mid 1 \leq \ell \leq m\} \subseteq L_{\log}.$$

Now for some $m \in \mathbb{N}$ and $1 \leq \ell \leq m$ we consider the word

$$z := [ri_{m,1}] \dots [ri_{m,\ell-1}][si_{m,\ell}][ri_{m,\ell+1}] \dots [ri_{m,2m-1}] \in S_m.$$

Corresponding to Ogden's Lemma we mark all the 0's in the ℓ -th block. Then we can write $z = uvwxy$ such that for a nonterminal A we have $S \Rightarrow_G^* uAv$, $A \Rightarrow_G^* vAx$ and $A \Rightarrow_G^* w$. By the iteration theorem v or x lie completely inside the 0's of block ℓ . Assume v lies completely in the 0's of block ℓ and $|x|_1 > 0$. Now $|x|_1$ is even, because otherwise by pumping only once we would obtain a word with an even number of blocks, which is impossible by the definition of the language. But then after pumping up $m + 3$ times we obtain a word which has enough repeated occurrences of a sequence of $2n$ blocks for some $n \in \mathbb{N}$, such that the condition of Lemma 2 is satisfied. Thus x cannot contain 1's in this case. The case that x lies completely in block ℓ and $|v|_1 > 0$ is treated analogously. Hence x and v cannot contain 1's. Thus both x and v lie completely in the 0's of one block, respectively. Assume x and v do not lie in the same block and $x \neq \varepsilon$ and $v \neq \varepsilon$. That is, block ℓ can be pumped together with a block ℓ' . Assume $\ell' \leq m$ then after one pumping step we obtain a word with two isolated blocks, which is a contradiction. Assume $\ell' > m$ then after one pumping step we obtain a word with a forward crack in block $2m - \ell'$ and a backward crack in block $2m - \ell' + 1$ again a contradiction. Note that in both blocks the correlation is either destroyed if it held before, or its partner is block ℓ and then due to the choice of s and r the crack is not repaired by one pumping step. Hence x and v either both lie inside block ℓ or the one which doesn't is the empty word.

Thus only block ℓ is pumped up. And by repeated pumping we can repair the cracks in block ℓ and obtain $(r * z_m)$. That is, all the words in S_m can be pumped up to yield $(r * z_m)$. Now assume that among the derivation trees obtained by this method there are two which are equal. Then we can pump two different

blocks $1 \leq \ell_1, \ell_2 \leq m$ independently. Thus by pumping once in both blocks we obtain a word with two isolated blocks, which is a contradiction.

Finally we have proved that $(r * z_m)$ has at least m derivation trees. Now the length of $(r * z_m)$ increases exponentially with respect to m . Hence the ambiguity is logarithmic with respect to the length of the word.

The proof that $L_{\sqrt{\cdot}}$ is inherently square-root ambiguous is analogous.

4 Conclusion

Here we have presented first examples of linear context-free languages with non-constant sublinear ambiguity. By concatenation we can get some other sublinear ambiguities. Is it possible to find nonconstant sublogarithmic ambiguity? Can we characterize the possible complexity classes? These questions are deeply connected with the structure of the intersection of context-free languages. To see this we consider the languages $L_1 := \{1^i 0^{2^i} \mid i \in \mathbb{N}\}$ and $L_2 := \{0^i 1^{2^i} \mid i \in \mathbb{N}\}$. Now we define the unambiguous languages $L'_1 := 0L_1^*$ and $L'_2 := L_2^*0^*$. The language $L'_1 \cap L'_2$ contains only $\mathcal{O}(\log n)$ words with a length up to n . Of course $L'_1 \cup L'_2$ has the degree of ambiguity 2, but ambiguity is “needed” only logarithmic many times. The languages L'_1 and L'_2 are slightly modified versions of languages found in [3]. The main question was how sublinear “density” of the intersection can be transformed into an inherent degree of ambiguity. The idea was to concatenate L_1^* and L_2^* buffered with a free block to interconnect the correlations and hide the factorization. This led to the (non-linear) language $L_1^* 1^+ L_2^*$ which is a context-free language with inherent logarithmic ambiguity.

Recall that intersections of context-free languages can have a very complicated structure. If we denote the set of computations of a Turing machine M by sequences of configurations, where every second configuration is written in reverse, then we obtain the set of valid computations. In [2, Lemma 8.6] it is shown that this set is the intersection of two linear languages.

Thus if our method of transforming the “density” of an intersection into an inherent degree of ambiguity can be generalized, we can hope for a variety of sublinear ambiguities.

Acknowledgements. Thanks to Prof. Dr. Friedrich Otto, Dr. Dieter Hofbauer, and Gundula Niemann for proofreading, valuable discussions and L^AT_EX tips.

References

1. J. Berstel. *Transductions and Context-Free Languages*. Teubner, 1979.
2. J.E. Hopcroft, J.D. Ullman. *Introduction to Automata Theory, Formal Languages, and Computation*. Addison-Wesley, 1979.
3. R. Kemp. A Note on the Density of Inherently Ambiguous Context-free Languages. *Acta Informatica* 14, pp. 295–298, 1980.

4. M. Naji. *Grad der Mehrdeutigkeit kontextfreier Grammatiken und Sprachen*. Diplomarbeit, FB Informatik, Johann–Wolfgang–Goethe–Universität, Frankfurt am Main, 1998.
5. K. Wich. *Kriterien für die Mehrdeutigkeit kontextfreier Grammatiken*. Diplomarbeit, FB Informatik, Johann–Wolfgang–Goethe–Universität, Frankfurt am Main, 1997.
6. K. Wich. Exponential Ambiguity of Context-free Grammars. *Proc. 4th Int. Conf. on Developments in Language Theory '99*, World Scientific, Singapore, to appear.

An Automata-Based Recognition Algorithm for Semi-extended Regular Expressions

Hiroaki Yamamoto

Department of Information Engineering, Shinshu University,
4-17-1 Wakasato, Nagano-shi, 380-8553 Japan.
yamamoto@cs.shinshu-u.ac.jp

Abstract. This paper is concerned with the recognition problem for semi-extended regular expressions: given a semi-extended regular expression r of length m and an input string x of length n , determine if $x \in L(r)$, where $L(r)$ denotes the language denoted by r . Although the recognition algorithm based on nondeterministic finite automata (NFAs) for regular expressions is widely known, a similar algorithm based on finite automata is currently not known for semi-extended regular expressions. The existing algorithm is based on dynamic programming. We here present an efficient automata-based recognition algorithm by introducing a new model of alternating finite automata called partially input-synchronized alternating finite automata (PISAFAs for short). Our algorithm based on PISAFAs runs in $O(mn^2)$ time and $O(mn + kn^2)$ space, though the existing algorithm based on dynamic programming runs in $O(mn^3)$ time and $O(mn^2)$ space, where k is the number of intersection operators occurring in r . Thus our algorithm significantly improves the existing one.

1 Introduction

This paper is concerned with the recognition problem for semi-extended regular expressions (that is, regular expressions with intersection). Given a semi-extended regular expression r of length m and an input string x of length n , the recognition problem is to determine if $x \in L(r)$, where $L(r)$ denotes the language denoted by r . It is widely known that such a recognition problem can be applied to the pattern matching problem for semi-extended regular expressions [1,4,8]. The standard recognition algorithm for regular expressions runs in $O(mn)$ time and $O(m)$ space, based on nondeterministic finite automata (NFAs for short) [1,3,6]. Myers [8] has improved this algorithm and has given an $O(mn/\log n)$ time and space recognition algorithm. Furthermore, an algorithm based on deterministic finite automata is also shown in [1]. Thus, for regular expressions, several recognition algorithms based on finite automata have been given, but any efficient algorithm based on finite automata is not known for semi-extended regular expressions. Although semi-extended regular expressions also denote only regular sets, they shorten the length of the expressions needed to describe certain regular sets. It is, therefore, important to design an efficient recognition algorithm for semi-extended regular expressions. When we try to

translate semi-extended regular expressions to NFAs in the standard way, intersection requires a multiplicative increase in the number of states. Since operators can be nested, the number of states exponentially increases. This suggests that any algorithm which uses this translation as one of its step for semi-extended regular expressions is going to be an exponential-time algorithm, and hence another approach has been taken. For example, as seen in [6], the existing algorithm uses a dynamic programming technique. The aim of this paper is to show that we can design an efficient automata-based recognition algorithm for semi-extended regular expressions.

Chandra et al. [2] introduced alternating finite automata (AFAs for short) as a generalization of NFAs and showed that AFAs also exactly accept regular sets. Although universal states of AFAs seem to correspond to intersection of semi-extended regular expressions, it is difficult to use AFAs for our aim. Slobodova [9] introduced synchronized alternating finite automata (SAFAs for short), which are an extension of AFAs, and showed that SAFAs can accept a wider class of languages than the class of regular sets. Hromkovic et al. [7] improved the result and showed that the class of languages accepted by one-way SAFAs is exactly equal to the class of context-sensitive languages. Thus SAFAs are more powerful than AFAs, though it is also difficult to use SAFAs. Recently, Yamamoto [10] introduced a new notion of synchronization called input-synchronization, and studied the power of input-synchronized alternating finite automata. The input-synchronization seems to be suitable for our aim.

In this paper, we will introduce a new model of alternating finite automata, called partially input-synchronized alternating finite automata (PISAFAs for short), and will show an efficient recognition algorithm based on PISAFAs for semi-extended regular expressions. PISAFAs are a variant of input-synchronized alternating finite automata for designing the recognition algorithm. In addition, PISAFAs are also a generalization of AFAs because a PISAFa without any synchronizing state is just an AFA.

Now let us give the definition of semi-extended regular expressions to recall it.

Definition 1. *Let Σ be an alphabet. The semi-extended regular expressions over Σ are defined as follows.*

1. $\emptyset$, ϵ and a ($\in \Sigma$) are semi-extended regular expressions that denote the empty set, the set $\{\epsilon\}$ and the set $\{a\}$, respectively.
2. Let r_1 and r_2 be semi-extended regular expressions denoting the sets R_1 and R_2 , respectively. Then $(r_1 \vee r_2)$, $(r_1 r_2)$, (r_1^*) and $(r_1 \wedge r_2)$ are also semi-extended regular expressions that denote the sets $R_1 \cup R_2$, $R_1 R_2$, R_1^* and $R_1 \cap R_2$, respectively.

Our main result is as follows:

- Let r be a semi-extended regular expression of length m with k intersection operators ($\wedge$ -operators), and let x be an input string of length n . Then we present an $O(mn^2)$ -time and $O(mn + kn^2)$ -space algorithm which determines if $x \in L(r)$.

For extended regular expressions (that is, semi-extended regular expressions with complement), the algorithm based on a dynamic programming technique is known and it runs in $O(mn^3)$ time and $O(mn^2)$ space (see [6]). It is clear that this algorithm can solve the recognition problem for semi-extended regular expressions within the same complexities, but it has never been known whether or not the complexities can be improved. Hence our algorithm significantly improves the existing one for semi-extended regular expressions, especially for the time complexity. In addition, our algorithm agrees with the standard recognition algorithm based on NFAs if r is a regular expression. Thus our result says that automata-theoretic techniques are applicable for semi-extended regular expressions as well as regular expressions.

2 Partially Input-Synchronized Alternating Finite Automata

We first define partially input-synchronized alternating finite automata (PISAFAs for short).

Definition 2. A PISAFa M is an eight-tuple $M = (Q, S, \Sigma, \delta, q_0, \mu, \psi, F)$, where

- Q is a finite set of states,
- $S (\subseteq Q)$ is a finite set of synchronizing states,
- Σ is a finite input alphabet,
- $q_0 (\in Q)$ is the initial state,
- μ is a function mapping Q to $\{\vee, \wedge\}$,
- ψ is a function mapping S to $S \cup \{\perp\}$, called a parent function,
- $F (\subseteq Q)$ is a set of final states,
- δ is a transition function mapping $Q \times (\Sigma \cup \{\epsilon\})$ to 2^Q , where ϵ denotes the empty string.

If $\mu(q) = \wedge$ ($\vee$, respectively), then q is called a *universal state* (an *existential state*, respectively). A configuration of M is defined to be a pair (q, pos) , where q is a state and pos is a position of the input head. If q is a universal (existential, respectively) state, then (q, pos) is called a *universal configuration* (an *existential configuration*, respectively). Among these configurations, if a state q is in F , then the configuration is also called an *accepting configuration*. The configuration $(q_0, 1)$ is called an *initial configuration*. For any synchronizing states $p, q \in S$, p is called a *parent* of q if $\psi(q) = p$. If $\psi(q) = \perp$, then q does not have any parent. The interpretation of $\delta(q, a) = \{q_1, \dots, q_l\}$ is that M reads the input symbol a and changes the state from q to $q_1, \dots, q_l$. This time, if $a \neq \epsilon$, then M advances the input head one symbol right, and if $a = \epsilon$, then M does not advance the input head (this is called an ϵ -move). We give the precise definition of an accepting computation of a PISAFa.

Definition 3. A full computation tree of a PISAFa M on an input $x = a_1 \cdots a_n$ is a labelled tree T such that

- each node of T is labelled with a configuration of M ,
- each edge of T is labelled with a symbol in $\{a_1, \dots, a_n, \epsilon\}$,
- the root of T is labelled with $(q_0, 1)$,
- if a node v of T is labelled with (q, pos) and for a symbol $a \in \{\epsilon, a_{\text{pos}}\}$, $\delta(q, a) = \{q_1, \dots, q_k\}$ is defined, then v has k children $v_1, \dots, v_k$ such that each v_i is labelled with (q_i, pos') and every edge e_i from v to v_i is labelled with the symbol a . Furthermore, if $a = a_{\text{pos}}$, then $\text{pos}' = \text{pos} + 1$, and if $a = \epsilon$, then $\text{pos}' = \text{pos}$.

Definition 4. Let T be a full computation tree of a PISAFM M on an input x of length n and let v_0 be the root of T . For a node v of T , let $\alpha = (p_1, b_1) \cdots (p_u, b_u)$ be the maximum sequence of labels on the path from v_0 to v satisfying the following: (1) $1 \leq b_1 \leq b_2 \leq \dots \leq b_u \leq n$, and (2) for any i ($1 \leq i \leq u$), p_i is a synchronizing state. Then the sequence α is called a synchronizing sequence of v . In addition, for any synchronizing state q_s with $\psi(q_s) = p$, we divide α into subsequences $\alpha_1, \dots, \alpha_e$ by p such that (1) $\alpha = \alpha_1 \cdots \alpha_e$, (2) for any $1 \leq l \leq e - 1$, α_l always ends with a label (p, b) in which b is any position, (3) for any $1 \leq l \leq e$, there exist no labels with p in α_l except the last one. Note that, if $\psi(q_s) = \perp$, then $\alpha_1 = \alpha$. For each α_l , let us consider the subsequence $\beta_l = (q_s, b_{l_1}) \cdots (q_s, b_{l_2})$ which is made from α_l by picking up all (p_j, b_j) with $p_j = q_s$. Then, we call $\langle \beta_1, \dots, \beta_e \rangle$ a q_s -synchronizing sequence of v . For example, let $\alpha = (p_1, 2)(p_2, 4)(p_2, 6)(p_1, 8)(p_1, 9)(p_2, 10)(p_3, 12)$ be a synchronizing sequence of v . Let us focus on p_1 . If $\psi(p_1) = p_2$, then α is divided into four subsequences $(p_1, 2)(p_2, 4)$, $(p_2, 6)$, $(p_1, 8)(p_1, 9)(p_2, 10)$, and $(p_3, 12)$. Hence we have $\langle (p_1, 2), \epsilon, (p_1, 8)(p_1, 9), \epsilon \rangle$ as a p_1 -synchronizing sequence of v .

Definition 5. A computation tree T' of a PISAFM M on an input x of length n is a subtree of a full computation tree T such that

- if v is labelled with a universal configuration, then v has the same children as in T ,
- if v is labelled with an existential configuration, then v has at most one child,
- let v_1 and v_2 be arbitrary nodes. For any synchronizing state q_s , let $\langle \beta_1, \dots, \beta_{l_1} \rangle$ and $\langle \gamma_1, \dots, \gamma_{l_2} \rangle$ ($l_1 \leq l_2$) be q_s -synchronizing sequences of v_1 and v_2 , respectively. Then for any i ($1 \leq i \leq l_1$), β_i is an initial subsequence of γ_i or vice versa. This condition means that two processes read input symbols on the same positions at the synchronizing state q_s until at least one process encounters a parent of q_s . For example, two q_s -synchronizing sequences $\langle (q_s, 2), \epsilon, (q_s, 8)(q_s, 9), \epsilon \rangle$ and $\langle \epsilon, (q_s, 3), (q_s, 8)(q_s, 9)(q_s, 10), (q_s, 11) \rangle$ satisfy this condition. We call this condition partially input-synchronization.

Definition 6. An accepting computation tree of a PISAFM M on an input x of length n is a finite computation tree of M on x such that each leaf is labelled with an accepting configuration with the input head position $n + 1$, that is, labelled with a label $(q, n + 1)$ with $q \in F$.

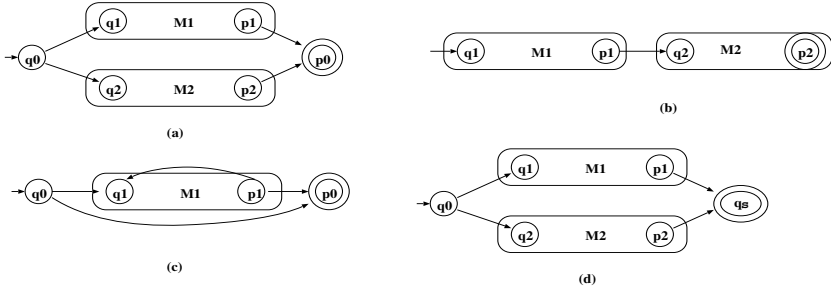


Fig. 1. Constructions in Theorem 1. (a) For union. (b) For concatenation. (c) For closure. (d) For intersection

We say that a PISAFAs M accepts an input x if there exists an accepting computation tree of M on x such that the root is labelled with the initial configuration. We denote the language accepted by M by $L(M)$.

3 Linear Translation from Semi-extended Regular Expressions to PISAFAs

In this section, we show that a semi-extended regular expression of length m can be translated to an $O(m)$ -state PISAFAs.

Theorem 1. *Let r be a semi-extended regular expression of length m . Then we can construct a PISAFAs M such that M has at most $O(m)$ states and accepts the language denoted by r .*

Proof. Sketch. The algorithm to construct M from a semi-extended regular expression r can be designed by using the same technique as the translation from regular expressions to NFAs with ϵ -moves except for intersection. Namely, the construction of M is given by induction on the number of operators in the semi-extended regular expression r . Fig. 1 shows the outline of the construction, and (a), (b), (c) and (d) depict $r = r_1 \vee r_2$ (union), $r = r_1 r_2$ (concatenation), $r = r_1^*$ (closure), and $r = r_1 \wedge r_2$ (intersection), respectively. Here M_1 and M_2 are PISAFAs for r_1 and r_2 , respectively. In (d), note that the initial state q_0 is universal, and the final state q_s is synchronizing. The parent function ψ of M is defined as follows. Let ψ_1 and ψ_2 be the parent functions of M_1 and M_2 , respectively, and let q be any synchronizing state in M_1 or M_2 . For (a), (b) and (c), $\psi(q) = \psi_1(q)$ if q is in M_1 , and $\psi(q) = \psi_2(q)$ if q is in M_2 . For (d), $\psi(q_s) = \perp$, and if q is in M_1 and $\psi_1(q) = \perp$ (q is in M_2 and $\psi_2(q) = \perp$, respectively), then $\psi(q) = q_s$; otherwise $\psi(q) = \psi_1(q)$ ($\psi(q) = \psi_2(q)$, respectively).

The correctness of the translation can be proved by induction on the number of operators. The difference from regular expressions is that there exist intersection operators. Since M_1 and M_2 exactly accept $L(r_1)$ and $L(r_2)$, respectively,

in the cases $r = r_1 \vee r_2$ and $r = r_1 \wedge r_2$, it is clear that M accepts $L(r)$. The case $r = r_1 r_2$ has a difficulty when $r_1 = r_1^1 \wedge r_1^2$, that is, the machine M_1 must accept the same string for r_1^1 and r_1^2 . However, this difficulty is solved by noting that M_1 has always a synchronizing state corresponding to the operator $\wedge$ between r_1^1 and r_1^2 . Namely, the synchronizing state forces M_1 to accept the same string. Hence M accepts $L(r)$. The case $r = r_1^*$ also has the similar difficulty. However, this can be also overcome by the synchronizing state of M_1 similarly. $\square$

4 Recognition Algorithm for Semi-extended Regular Expressions

In this section, we will present an $O(mn^2)$ time and $O(mn + kn^2)$ space recognition algorithm based on PISAFAs for a semi-extended regular expression r of length m and an input string x of length n . Here k is the number of intersection operators occurring in r . We first give the outline of the algorithm and then give the detail.

4.1 Outline of the Algorithm

Before describing the detail of a recognition algorithm, we here give the outline. Our algorithm becomes an extension of the algorithm based on NFAs for regular expressions, but it is not a straightforward extension. Let r be a semi-extended regular expression and let M be a PISAFa obtained by the linear translation in Theorem 1. The main part of our algorithm is to simulate M on an input $x = a_1 \cdots a_n$. Note that our algorithm is intended to simulate a PISAFa obtained by the linear translation but not any PISAFAs. Such PISAFAs have a restricted structure as follows.

Property 1 For any state q , the number of transitions from q is at most two.

Property 2 For any state q , all the transitions from q is done by the same symbol $a \in \Sigma \cup \{\epsilon\}$. If $a \in \Sigma$, then the number of transitions is exactly one.

Property 3 For any universal state q , there exists just one synchronizing state q_s corresponding to q such that all the computations starting from q always visit q_s on the way to the accepting state.

To simulate M , we introduce a set U_p called an *existential-element set*, where p forms either q or q^i for any state q of M and any $1 \leq i \leq n$. The elements of U_p are classified into two kinds of elements. One is a state of M , and the other is a pair of states called a *universal element*. Simply speaking, U_p keeps states reachable from the state denoted by p . If M does not have any universal states, then we can simulate M using just one existential-element set U_{q_0} , where q_0 is the initial state of M . Note that, in this case, our algorithm agrees with the existing algorithm based on NFAs. Now let us consider the general case in which M has universal state and synchronizing states. Our simulation is to construct a *directed computation graph* $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ consisting of existential-element sets such

that (1) $\mathcal{U}$ is the set of *nodes*, which consists of existential-element sets, and $\mathcal{E}$ is the set of *edges*, which consists of pairs (U, U') of nodes, (2) U_{q_0} is called a *source node*, (3) let U_p, U_{p_1}, U_{p_2} be nodes of $\mathcal{U}$. Then U_p contains a universal element $p_1 p_2$ if and only if there exist directed edges (U_p, U_{p_1}) and (U_p, U_{p_2}) in $\mathcal{E}$.

A node U_p of $\mathcal{G}$ is called an *accepting node* if U_p satisfies at least one of the following (1) and (2): (1) U_p contains a semi-accepting state q , where q is said to be *semi-accepting* if and only if there is an accepting computation tree M such that the root is labelled with $(q, 1)$ and all edges are labelled with the empty string ϵ , (2) U_p contains a universal element $p_1 p_2$ such that both U_{p_1} and U_{p_2} are accepting. A directed computation graph $\mathcal{G}$ is said to be *accepting* if the source node U_{q_0} of $\mathcal{G}$ is accepting.

The simulation starts with $\mathcal{U} = \{U_{q_0}\}$ and $\mathcal{E} = \emptyset$, where $U_{q_0} = \{q_0\}$. Let $\mathcal{G}_{i-1} = (\mathcal{U}_{i-1}, \mathcal{E}_{i-1})$ be a directed computation graph obtained after processing $a_1 \cdots a_{i-1}$. Note that $\mathcal{G}_{i-1}$ satisfies the property that for any $U_p \in \mathcal{U}_{i-1}$, $q \in U_p$ if and only if M can reach the state q from the state denoted by p by $a_1 \cdots a_{i-1}$. Then $\mathcal{G}_i = (\mathcal{U}_i, \mathcal{E}_i)$ is built as follows. First, for any existential-element set $U_p \in \mathcal{U}_{i-1}$ and any state $q \in U_p$, we compute every state q' reachable existentially from q by ϵ -moves. We simulate the computation by ϵ -moves using two functions, *EpsilonMove* and *SyncEpsilon*, to facilitate a check on the partially input-synchronization. The function *EpsilonMove* simulates ϵ -moves from q to a synchronizing state, and the function *SyncEpsilon* simulates one ϵ -move from a synchronizing state. During the simulation, if q' is existential, then we add it to U_p . If q' is universal and has a transition $\delta(q', \epsilon) = \{q_1, q_2\}$, then we add a universal element $q_1^i q_2^i$ to U_p , add two nodes $U_{q_1^i} = \{q_1\}$ and $U_{q_2^i} = \{q_2\}$ to $\mathcal{U}_{i-1}$, and add two edges $(U_p, U_{q_1^i})$ and $(U_p, U_{q_2^i})$ to $\mathcal{E}_{i-1}$. The partially input-synchronization is checked after *EpsilonMove* as follows. Let U_p, U_{p_1} , and U_{p_2} be nodes such that both (U_p, U_{p_1}) and (U_p, U_{p_2}) are in $\mathcal{E}_{i-1}$. This time, if these two nodes have the same synchronizing state q_s during the simulation (this means that two processes read a symbol on the same position in q_s), then q_s is removed from both U_{p_1} and U_{p_2} , and is added to U_p . After computing all states reachable by ϵ -moves, we compute only states reachable from states belonging to a set of $\mathcal{U}_{i-1}$ on a_i , and finally obtain $\mathcal{G}_i = (\mathcal{U}_i, \mathcal{E}_i)$. The above process is repeatedly performed from a_1 to a_n . Let $\mathcal{G}_n$ be the directed computation graph obtained after processing a_n . In order to determine if x is accepted by M , we check whether the source node U_{q_0} is accepting or not. If U_{q_0} is accepting, then our algorithm accepts x ; otherwise reject x . The time and space complexities mainly depend on the size of a directed computation graph. Therefore, we design the algorithm so that we do not increase the number of nodes and the number of edges as much as possible.

4.2 Algorithm in Detail

Now let us give the detail of the algorithm below. Given a semi-extended regular expression r and an input string x , the computation starts with the following ACCEPT.

Algorithm ACCEPT(r, x)

Input: A semi-extended regular expression r , an input string x .

Output: If $x \in L(r)$, then return *YES*; otherwise return *NO*.

Step 1. Translate r to a PISFA $M = (Q, S, \Sigma, \delta, q_0, \mu, \psi, F)$ using the linear translation in Theorem 1.

Step 2. Execute $\text{SIMULATE}(M, x, q_0, F)$, and if it returns *YES*, then output *YES*; otherwise output *NO*.

Function SIMULATE(M, x, q, F)

Input: A PISFA M derived by the linear translation, a string $x = a_1 \cdots a_n$, a state q , and a set F of final states.

Output: If M accepts x , then return *YES*; otherwise *NO*.

Comment: This function directly simulates M starting in the state q . The simulation constructs a directed computation graph $\mathcal{G}$.

Step 1. Initialization. Set $\mathcal{G} = (\mathcal{U}, \mathcal{E})$, where $\mathcal{U} = \{U_q\}$, $U_q := \{q\}$, and $\mathcal{E} = \emptyset$. In addition, $\text{Sync} := \emptyset$.

Step 2. $F_{\text{accept}} := \text{AcceptState}(F)$.

Step 3. For $i = 1$ to n , do the following:

1. $\mathcal{G}_{\text{old}} := (\emptyset, \emptyset)$.
2. while $\mathcal{G}_{\text{old}} \neq \mathcal{G}$ do the following:
 - a) $\mathcal{G}_{\text{old}} := \mathcal{G}$.
 - b) $\mathcal{G} := \text{EpsilonMove}(\mathcal{G}, i)$.
 - c) $\mathcal{G} := \text{SyncCheck}(\mathcal{G})$.
 - d) $\mathcal{G} := \text{SyncEpsilon}(\mathcal{G})$.
3. $\mathcal{G} := \text{GoTo}(\mathcal{G}, a_i)$.

Step 4. If $\text{AcceptCheck}(\mathcal{G}, U_q, F_{\text{accept}})$ returns *YES*, then return *YES*; otherwise return *NO*.

Function AcceptState(F)

Input: A set F of final states.

Output: A set F_1 of semi-accepting states.

Step 1. $F_1 := F$, $F_{\text{new}} := F_1$, $F_{\text{tmp}} := \emptyset$ and $F_1^{\text{old}} := \emptyset$.

Step 2. $F' := \emptyset$, and for all states q , $F_q := \emptyset$.

Step 3. while $F_1^{\text{old}} \neq F_1$ do the following:

1. $F_1^{\text{old}} := F_1$.
2. For all $q \in Q - F_1$ such that $\delta(q, \epsilon) \cap F_{\text{new}} \neq \emptyset$, do the following:
 - a) If q is existential and there exists $q_1 \in \delta(q, \epsilon)$ such that $q_1 \in F_1$, then $F_{\text{tmp}} := F_{\text{tmp}} \cup \{q\}$.
 - b) If q is universal and both elements of $\delta(q, \epsilon)$ are in F_1 , then $F_{\text{tmp}} := F_{\text{tmp}} \cup \{q\}$.
3. $F_1 := F_1 \cup F_{\text{tmp}}$, $F_{\text{new}} := F_{\text{tmp}}$ and $F_{\text{tmp}} := \emptyset$.

Step 3. Return F_1 .

Function EpsilonMove($\mathcal{G}, i$)

Input: A directed computation graph $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ and an input position i .

Output: A directed computation graph $\mathcal{G}' = (\mathcal{U}', \mathcal{E}')$.

Comment: For any $U_p \in \mathcal{U}$ and any non-synchronizing state $q \in U_p$, this function computes states reachable from q using ϵ -moves only.

Step 1. For all $U_p \in \mathcal{U}$, do the following:

1. $U_{old} := \emptyset$.
2. while $U_{old} \neq U_p$ do the following;
 - a) $U_{old} := U_p$.
 - b) For all $q \in U_p$ such that $q \in Q - S$ and $\delta(q, \epsilon) \neq \emptyset$, do the following:
 - i. If q is existential, then $U_p := U_p \cup \delta(q, \epsilon)$.
 - ii. If q is universal, then do the following: Here let $\delta(q, \epsilon) = \{q_1, q_2\}$ and let q_s be the synchronizing state corresponding to q .
 - A. if both $U_{q_1^i}$ and $U_{q_2^i}$ are already in $\mathcal{U}$, then $U_p := U_p \cup \{q_1^i, q_2^i\}$ and $\mathcal{E} := \mathcal{E} \cup \{(U_p, U_{q_1^i}), (U_p, U_{q_2^i})\}$.
 - B. if both $U_{q_1^i}$ and $U_{q_2^i}$ are not in $\mathcal{U}$ yet, then $U_p := U_p \cup \{q_1^i, q_2^i\}$, $U_{q_1^i} := \{q_1\}$ and $U_{q_2^i} := \{q_2\}$, and then $(\mathcal{U}_1, \mathcal{E}_1) := \text{EpsilonMove}(\{U_{q_1^i}\}, \emptyset, i)$ and $(\mathcal{U}_2, \mathcal{E}_2) := \text{EpsilonMove}(\{U_{q_2^i}\}, \emptyset, i)$. After that, $\mathcal{U} := \mathcal{U} \cup \mathcal{U}_1 \cup \mathcal{U}_2$, $\mathcal{E} := \mathcal{E} \cup \mathcal{E}_1 \cup \mathcal{E}_2 \cup \{(U_p, U_{q_1^i}), (U_p, U_{q_2^i})\}$, and $\text{Sync} := \text{Sync} \cup \{(q_s, U_{q_1^i}, U_{q_2^i})\}$.
 - iii. The state q is marked in order to show that q has already been processed.

Step 3. Return $\mathcal{G} = (\mathcal{U}, \mathcal{E})$.

Function SyncCheck($\mathcal{G}$)

Input: A directed computation graph $\mathcal{G} = (\mathcal{U}, \mathcal{E})$.

Output: A directed computation graph $\mathcal{G}' = (\mathcal{U}', \mathcal{E}')$.

Comment: For any triple (q, U_{p_1}, U_{p_2}) in Sync , this function checks whether or not U_{p_1} and U_{p_2} satisfy the input-synchronization in the state q .

Step 1. For all $(q, U_{p_1}, U_{p_2}) \in \text{Sync}$, do the following:

1. If $q \in U_{p_1}$ and $q \in U_{p_2}$,
 - a) $U_{p_1} := U_{p_1} - \{q\}$, $U_{p_2} := U_{p_2} - \{q\}$.
 - b) For all U_p such that both edges (U_p, U_{p_1}) and (U_p, U_{p_2}) are in $\mathcal{E}$, $U_p := U_p \cup \{q\}$.
 - c) Remove (q, U_{p_1}, U_{p_2}) from Sync .

Step 2. Return $\mathcal{G} = (\mathcal{U}, \mathcal{E})$.

Function SyncEpsilon($\mathcal{G}$)

Input: A directed computation graph $\mathcal{G} = (\mathcal{U}, \mathcal{E})$.

Output: A directed computation graph $\mathcal{G}' = (\mathcal{U}', \mathcal{E}')$.

Comment: For any $U_p \in \mathcal{U}$ and any synchronizing state $q_s \in \text{Sync}$, this function computes states reachable from q_s by just one ϵ -move.

Step 1. For all $q_s \in \text{Sync}$ and all $U_p \in \mathcal{U}$, $U_p := U_p \cup \delta(q_s, \epsilon)$.

Step 2. Return $\mathcal{G} = (\mathcal{U}, \mathcal{E})$.

Function GoTo($\mathcal{G}, a$)

Input: A directed computation graph $\mathcal{G} = (\mathcal{U}, \mathcal{E})$ and an input symbol a .

Output: A directed computation graph $\mathcal{G}' = (\mathcal{U}', \mathcal{E}')$.

Comment: For any $U_p \in \mathcal{U}$ and any state $q \in U_p$, if $\delta(q, a) \neq \emptyset$, then compute the next state. If $\delta(q, a) = \emptyset$, then q is removed from U_p . At the moment, if $(q, U_{p_1}, U_{p_2}) \in \text{Sync}$, then it means that M does not satisfy the input-synchronization in the state q because either U_{p_1} or U_{p_2} only has a synchronizing state q . Hence such a state q is first removed from U_{p_1} and U_{p_2} .

Step 1. For all $(q, U_{p_1}, U_{p_2}) \in \text{Sync}$, if $q \in U_{p_1}$ then $U_{p_1} := U_{p_1} - \{q\}$, and if $q \in U_{p_2}$ then $U_{p_2} := U_{p_2} - \{q\}$.

Step 2. For all $U_p \in \mathcal{U}$, do the following:

1. For all $q \in U_p$, do the following:

a) If $\delta(q, a) = \{q'\}$, then $U_p := (U_p - \{q\}) \cup \{q'\}$,

b) If $\delta(q, a) = \emptyset$, then $U_p := U_p - \{q\}$.

Step 3. $\text{Sync} := \emptyset$, and return $\mathcal{G} = (\mathcal{U}, \mathcal{E})$.

Function AcceptCheck $(\mathcal{G}, U_p, F_1)$

Input: A directed computation graph $\mathcal{G} = (\mathcal{U}, \mathcal{E})$, an existential-element set $U_p \in \mathcal{U}$, and a set F_1 of semi-accepting states.

Output: If U_p is accepting, then this returns *YES*; otherwise *NO*.

Comment: This function checks whether or not the node U_p is accepting.

Step 1. If there exists a state $q \in U_p$ such that $q \in F_1$, then return *YES*.

Step 2. If there exists a universal element $q_1^i q_2^i \in U_p$ such that both $\text{AcceptCheck}(\mathcal{G}, U_{q_1^i}, F_1)$ and $\text{AcceptCheck}(\mathcal{G}, U_{q_2^i}, F_1)$ return *YES*, then return *YES*.

Step 3. Otherwise return *NO*.

The following theorem holds for the algorithm ACCEPT.

Theorem 2. *Given a semi-extended regular expression r of length m and an input string x of length n , the algorithm ACCEPT correctly determines if $x \in L(r)$ in $O(mn^2)$ time and $O(mn + kn^2)$ space, where k is the number of intersection operators occurring in r . In addition, if r is a regular expression, then ACCEPT runs in $O(mn)$ time and $O(m)$ space.*

References

1. A.V. Aho, Algorithms for finding patterns in strings, In J.V. Leeuwen, ed. Handbook of theoretical computer science, Elsevier Science Pub., 1990.
2. A.K. Chandra, D.C. Kozen and L.J. Stockmeyer, Alternation, J. Assoc. Comput. Mach. 28,1, 114-133, 1981.
3. C.H. Chang, and R. Paige, From regular expressions to DFA's using compressed NFA's, Theoret. Comput. Sci., 178, 1-36, 1997.
4. J.R. Knight and E.W. Myers, Super-Pattern matching, Algorithmica, 13, 1-2, 211-243, 1995.
5. J.Dassow, J.Hromkovic, J.Karhuaki, B.Rovan and A. Slobodova, On the power of synchronization in parallel computation, In Proc. 14th MFCS'89, LNCS 379,196-206, 1989.
6. J.E. Hopcroft and J.D. Ullman, Introduction to automata theory language and computation, Addison Wesley, Reading Mass, 1979.
7. J. Hromkovic, K. Inoue, B. Rován, A. Slobodova, I. Takanami and K.W. Wagner, On the power of one-way synchronized alternating machines with small space, International Journal of Foundations of Computer Science, 3, 1, 65-79, 1992.
8. G. Myers, A four Russians algorithm for regular expression pattern matching, J. Assoc. Comput. Mach. 39,4, 430-448, 1992.
9. A. Slobodova, On the power of communication in alternating machines, In Proc. 13th MFCS'88, LNCS 324,518-528, 1988.
10. H. Yamamoto, On the power of input-synchronized alternating finite automata, Proc. COCOON'2000, LNCS, to appear.

Author Index

- Ablayev, Farid 132
Abramsky, Samson 141
Ambos-Spies, Klaus 152

Barrière, Lali 162
Barrington, David. M. 172
Berstel, Jean 182
Biedl, Therese C. 192, 202
Blondel, Vincent D. 549
Boasson, Luc 182
Boer, Frank S. de 212
Bollig, Beate 222
Bonsangue, Marcello M. 212
Bouyer, P. 232
Brejová, Broňa 192
Buchholz, Thomas 243
Bultan, Tefvik 426

Caha, Rostislav 253
Carpi, Arturo 264
Carton, Olivier 275
Čenek, Eowyn 202
Chan, Timothy M. 202
Comellas, Francesc 285

Dal Zilio, Silvano 1
Dang, Zhe 426
Davenport, James H. 21
De Felice, Clelia 295
Demaine, Erik D. 202
Demaine, Martin L. 202
Demetrescu, Camil 36
Dezani-Ciancaglini, M. 304
Dobrev, Stefan 314
Dufourd, C. 232
Durand, Arnaud 323

Ebert, Todd 333
Ésik, Zoltán 343
Esparza, Javier 619

Fàbrega, Josep 162
Finkel, A. 353
Fleischer, Rudolf 202
Fleury, E. 232
Fotakis, D.A. 363

Gainutdinova, Aida 132
Gardner, Philippa 373
Gordon, Andrew D. 1
Gregor, Petr 253
Groote, Jan F. 383
Grosu, Radu 52

Hemaspaandra, Edith 64
Hemaspaandra, Lane A. 64, 394
Henriksen, Jesper G. 405
Hermann, Miki 323
Holzer, Markus 415
Honsell, F. 304

Ibarra, Oscar H. 426
Italiano, Giuseppe F. 36
Iwama, Kazuo 436

Jansen, Klaus 446

Kemmerer, Richard 426
Klein, Andreas 243
Klíma, Ondřej. 456
Kolaitis, Phokion G. 84, 323
Kosub, Sven 467
Kuich, Werner 488
Kumar, Narayan K. 405
Kupferman, Orna 497
Kutrib, Martin 243
Král, Daniel 477

Lafitte, Grégory 508
Lanotte, Ruggero 518
Leeuwen, Jan van 99
Lenisa, Marina 141
Lozin, Vadim V. 528
Luca, Aldo de 264
Ly, Olivier 539

Maggiolo-Schettini, Andrea 518
Mairesse, Jean 549
Mantaci, Sabrina 549
Maoz, Sharar 621
Matsuura, Akihiro 436
Mazoyer, Jacques 508

- McKenzie, Pierre 172, 415
 Mitjana, Margarida 285
 Montanari, Angelo 559
 Montanari, Ugo 569
 Moore, Cris 172
 Motohama, Y. 304
 Mukund, Madhavan 405
- Narayanan, Lata 285
 Nikolettseas, S.E. 363
- Ogihara, Mitsunori 394
 Opatrny, Jaroslav 285
- Papadopoulou, V.G. 363
 Paterson, Mike 436
 Peleg, David 579
 Petersen, Holger 589
 Petit, A. 232
 Pighizzini, Giovanni 599
 Pistore, Marco 569
 Pol, Jaco van de 383, 609
 Policriti, Alberto 559
 Porkolab, Lorant 446
 Prensa Nieto, Leonor 619
- Rabinovich, Alexander 629
 Reith, Steffen 640
- Savický, Petr 650
- Schwentick, Thomas 660
 Sieling, Detlef 650
 Slanina, Matteo 559
 Spirakis, P.G. 363
 Srba, Jiří 456
 Su, Jianwen 426
 Sutre, G. 353
- Tanaka, Miki 670
 Tesson, Pascal 172
 Thérien, Denis 172
 Thiagarajan, P.S. 405
 Thomas, Wolfgang 275
- Urvoy, Tanguy 680
- Vardi, Moshe Y. 84, 497
 Vinař, Tomáš 192
 Vollmer, Heribert 333, 640
- Wang, Ming-Wei 202
 Wechsung, Gerd 394
 Wich, Klaus 690
 Wiedermann, Jiří 99
 Wischik, Lucian 373
- Yamamoto, Hiroaki 699
- Zaks, Shmuel 114
 Zantema, Hans 609